MAGIS: LLM-Based Multi-Agent Framework for GitHub Issue ReSolution

Wei Tao
Fudan University
`wtao18@fudan.edu.cn`

Yucheng Zhou
University of Macau
`yucheng.zhou@connect.um.edu.mo`

Yanlin Wang
Sun Yat-sen University
`wangylin36@mail.sysu.edu.cn`

Wenqiang Zhang *
Fudan University
`wqzhang@fudan.edu.cn`

Hongyu Zhang
Chongqing University
`hyzhang@cqu.edu.cn`

Yu Cheng *
The Chinese University of Hong Kong
`chengyu@cse.cuhk.edu.hk`

Abstract

In software development, resolving the emergent issues within GitHub repositories is a complex challenge that involves not only the incorporation of new code but also the maintenance of existing code. Large Language Models (LLMs) have shown promise in code generation but face difficulties in resolving Github issues, particularly at the repository level. To overcome this challenge, we empirically study the reason why LLMs fail to resolve GitHub issues and analyze the major factors. Motivated by the empirical findings, we propose a novel LLM-based **M**ulti-**A**gent framework for **G**itHub **I**ssue re**S**olution, **MAGIS**, consisting of four agents customized for software evolution: Manager, Repository Custodian, Developer, and Quality Assurance Engineer agents. This framework leverages the collaboration of various agents in the planning and coding process to unlock the potential of LLMs to resolve GitHub issues. In experiments, we employ the SWE-bench benchmark to compare MAGIS with popular LLMs, including GPT-3.5, GPT-4, and Claude-2. MAGIS can resolve **13.94%** GitHub issues, significantly outperforming the baselines. Specifically, MAGIS achieves an eight-fold increase in resolved ratio over the direct application of GPT-4, the advanced LLM.

1 Introduction

In real-world software development, the code repository for a project is rarely set in stone. High-quality and popular software always evolves to address emergent bugs or new requirements. On platforms such as GitHub [21], issues typically signify the requirement for software evolution. However, addressing these issues poses significant challenges, as it requires implementing the code change across the entire repository and maintaining the existing functionality while integrating new capabilities. For example, `django`, a framework for over 1.6M projects has 34K issues [19]. Consequently, resolving GitHub issues remains a significant challenge across academia and industry [27, 5].

Large language models (LLMs) have demonstrated remarkable capabilities across a variety of tasks [8], including code generation and code understanding [64, 47]. Specifically, LLMs excel in

---

[*] Corresponding author

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

generating function-level code, as evidenced by their performance on numerous benchmark datasets such as MBPP [2] and HumanEval [12]. Despite their success, LLMs remain challenged in tasks that require advanced code generation capabilities, such as class-level code generation [14]. Moreover, LLMs exhibit limitations in processing excessively long context inputs and are subject to constraints regarding their input context length [33]. This limitation is particularly evident in repository-level coding tasks, such as solving GitHub issues, where the context comprises the entire repository, thus imposing constraints on directly using the full repository as input to LLMs.

To harness the full potential of LLMs, many LLM-based multi-agent systems are designed [23, 43, 52]. These methods have significantly improved LLMs' efficacy in code generation, enabling these systems to construct code repositories based on LLM. While these methods address the process of transitioning code repositories from inception to establishment, they rarely consider the handling of software evolution, e.g., resolving GitHub issues. For GitHub repositories, especially the popular ones, a large number of commits are pushed every day. These commits derive from a spectrum of evolutionary requirements that span bug fixes, feature additions, performance enhancements, etc [49]. For open-source software, new requirements frequently emerge as issues in the project's repository.

Recently, Jimenez et al. [27] developed a benchmark, namely SWE-bench, to investigate the capability of popular LLMs in addressing GitHub issues. Their study reveals that LLMs fail to resolve over 95% of instances, even when file paths that require modifications are provided. This significantly low rate underscores the importance of understanding the reasons behind their suboptimal performance.

In this study, we analyze the factors impacting the effectiveness of LLMs in resolving GitHub issues. Furthermore, our empirical analysis has concluded a correlation between locating files/lines to be modified and the performance of resolving GitHub issues. Based on these insights, we propose a novel LLM-based multi-agent framework, termed MAGIS, comprising four types of agents: Manager, Repository Custodian, Developer, and Quality Assurance (QA) Engineer. Our approach facilitates the resolution of GitHub issues through collaboration among agents, each fulfilling a unique role: the Manager coordinates the entire process, the Repository Custodian enhances locating files, the Developer performs code changes after locating lines, and the QA Engineer reviews the code change.

In our experiment, we evaluate our framework on SWE-bench and compare its performance against existing popular LLMs, such as ChatGPT-3.5 [37], GPT-4 [38], and Claude-2 [1]. The results demonstrate that our framework, utilizing GPT-4 as its base model, significantly outperforms baselines and achieves an eight-fold performance gain compared to the direct application of GPT-4. Further analysis reveals that additional factors, i.e., the planning of code change, locating lines within the code file, and code review process, can significantly influence the resolution rate.

Our main contributions are summarized as follows:

- We conduct an empirical analysis of LLMs in resolving GitHub issues and explore the correlation between locating code file/line, complexity of the code change, and the success rate in resolution.
- We propose a novel LLM-based multi-agent framework, MAGIS, to alleviate the limitations of existing LLMs on GitHub issue resolution. Both our designed four-type agents and their collaboration for planning and coding unlock LLMs' potential on the repository-level coding task.
- We compare our framework and other strong LLM competitors (i.e., GPT-3.5, GPT-4, and Claude-2) on the SWE-bench dataset. The results show MAGIS significantly outperforms these competitors. Further analysis confirms the effectiveness and necessity of our framework design.

2 Empirical Study

SWE-bench [27] reveals the challenges LLMs face in addressing GitHub issue resolution. For example, in their evaluation, GPT-4 can only resolve less than 2% issues of the test set. Conversely, in tasks like function-level code generation, LLMs exhibit superior performance (e.g., GPT-4 gets the score of 67.0 on HumanEval [36]). Given the complexity of GitHub issue resolution akin to repository-level coding, we aim to investigate **Why the Performance of Directly Using LLMs to Resolve GitHub Issue is Limited? (RQ 1)**. We answer this RQ from the following three aspects:

Locating the Files to be Modified. GitHub issue resolution is a repository-level coding task, distinguishing it from file-level coding tasks primarily in the challenge of locating the files requiring modification. Jimenez et al. [27] employ the BM25 method [45] to retrieve relevant code files that are subsequently utilized as input to the LLM. After employing retrieval methods, it is necessary to select the top-K files or truncate the content based on the maximum context length of the LLM. Incorporating more files can enhance recall scores. However, it also imposes significant demands on the capabilities of LLMs. As demonstrated by the study [27], Claude-2 exhibits a decrease in the resolved ratio (from 1.96% to 1.22%) as recall scores increase (from 29.58 to 51.06). This decline may be attributed to the inclusion of irrelevant files or the limited capacity of LLMs to process longer contexts effectively. Consequently, optimizing the performance of LLMs can be better achieved by striving for higher recall scores with a minimized set of files, thus suggesting a strategic balance between recall optimization and the number of chosen files.

Locating the Lines to be Modified. Beyond the impact of file locating, we delve into the generation of failed instances when the correct modified files were provided. A typical code change consists of multiple hunks, each specifying the line numbers targeted for modification and detailing the changes made at these locations. To quantitatively analyze the accuracy of line localization, we use the line numbers' range of the modified content in the reference code change as the basis assuming that the correct modification location of the code change is uniquely determined in most cases. By calculating the coverage ratio of the line number ranges of the generated and reference, we can estimate the accuracy of line localization in the generation process, i.e.,

$$\text{Coverage Ratio} = \frac{\sum_{i=0}^{n} \sum_{j=0}^{m} \left| [s_i, e_i] \cap [s'_j, e'_j] \right|}{\sum_{i=0}^{n} (e_i - s_i + 1)}, \tag{1}$$

where the numerator is the length of the intersection of modified lines between the reference divided into n hunks and the generation divided into m hunks, and the denominator is the number of modified lines in the reference. More details about Equation 1 can be found in Appendix A.1.

For 574 instances in the SWE-bench that experiments GPT-4 [27], the distribution of the coverage ratio between the results generated by three LLMs and the reference is shown in Fig. 1. From this, we observe that the performance of LLMs in generating the code change is probably related to their ability to locate code lines accurately (Detailed explanation can be found in Appendix A.2).

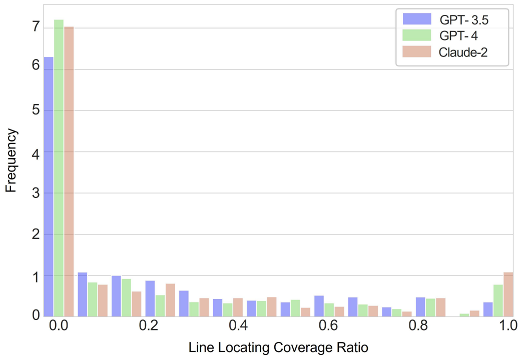


Figure 1: The comparison of line locating coverage ratio between three LLMs. The vertical axis representing the frequency of the range of line locating coverage ratio for each group, and the horizontal axis representing the coverage ratio.

Furthermore, we assess the relationship between the coverage ratio and the issue resolution by calculating their correlation coefficient. Given that the distribution of these variables exhibits skewness, and the resolution result is binary (resolved or not), logistic regression is employed for the analysis across three LLMs. However, due to the limited number of successfully generated instances on GPT-4 and GPT-3.5, a statistically significant relationship is only detected in the result generated by Claude-2. The result, i.e., P-value < 0.05, shows statistical significance. Specifically, with a coefficient, 0.5997, on Claude-2, there is a substantial and positive relation between improvements in the coverage ratio and the probability of successfully resolving issues, which demonstrates that locating lines is a key factor for GitHub issue resolution.

Complexity of the Code Changes. The complexity of the code change is reflected in various indices: the number of modified files, functions, hunks, and lines added or deleted. Firstly, we quantitatively assess the complexity by calculating the value of various indices corresponding to the reference code change. Secondly, the coefficient is calculated between the numbers in each index and the issue resolution. Tab. 1 shows the correlation scores under the logistic regression.

As shown in Tab. 1, all three LLMs demonstrate a statistically significant correlation with the issue resolution across several indices. The correlation scores for the number of files and functions modified

Table 1: Correlation between the complexity indices and the issue resolution.

LLM	# Files	# Functions	# Hunks	# Added LoC	# Deleted LoC	# Changed LoC
GPT-3.5	-17.57^*	-17.57^*	-0.06^*	-0.02	-0.03	-0.53^*
GPT-4	-25.15^*	-25.15^*	-0.06	-0.10	-0.04	-0.21
Claude-2	-1.47^*	-1.47^*	-0.11^*	-0.09^*	-0.07^*	-0.44^*

* The correlation between the index and the issue resolution is significant (P-value < 0.05).

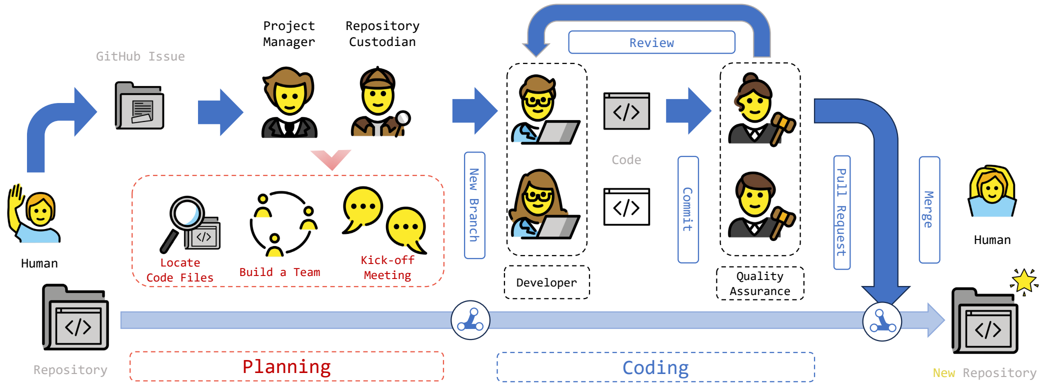


Figure 2: Overview of our framework, MAGIS. The detailed version can be found in Fig. 14.

are notably negative for all models, indicating that an increase in these indices is associated with a decreasing likelihood of issue resolution. This suggests that the more complex the code change, as indicated by a higher number of files and functions modified, may hinder the issue resolution. More analysis can be found in Appendix A.3. The analysis reveals a relationship between the complexity, as measured by several indices, and whether to successfully resolve the issues in software evolution. The negative correlations suggest that increased complexity, particularly in terms of the number of files and functions changed, tends to hinder issue resolution.

3 Methodology

Based on the empirical study identifying key factors affecting LLMs' issue resolution, we design the framework illustrated in Fig. 2. This framework aims to mitigate negative impacts by transforming the complex task of GitHub issue resolution into a collaborative effort. It incorporates four key roles for LLM-based agents working collaboratively in the workflow: ① *Manager*: this role tasks with team assembly, meeting organization, and plan formulation. ② *Repository Custodian*: it is responsible for locating the relevant files in the repository acording to the GitHub issue and recording the change of the repository. ③ *Developer*: this role participates in planning discussions and completes tasks from the Manager. ④ *Quality Assurance (QA) Engineer*: it reviews the code change from Developers to ensure the quality of the whole repository.

The collaborative process involves planning and coding. In the planning, an issue is assigned to the Manager and the Repository Custodian. The custodian identifies candidate files relevant to the issue for modification. With the issue description and a list of candidate files, the Manager defines tasks and assembles a team, where each member is a Developer specifically designed for the defined task. The Manager holds a kick-off meeting with Developers and devises a plan. During coding, Developers undertake their assigned tasks from the Manager, and the QA Engineer reviews each code change. If a change fails to meet quality standards, the QA Engineer provides feedback, prompting further revisions until the QA Engineer approves or a set iteration limit is reached. More details can be found in our GitHub repository [2].

---

[2] `https://github.com/co-evolve-lab/magis`

3.1 Agent Role Design

Our workflow draws inspiration from the GitHub Flow[22], an effective human workflow paradigm adopted by many software teams. Both the human workflow and our LLM-based agent framework prioritize collaboration among individuals with diverse skills. While the underlying principles are similar, there are notable differences. Accordingly, we have tailored the roles as follows:

- **Manager**. The Manager's role is pivotal in planning. In conventional setups, managers decompose the issue into tasks according to the pre-formed team and allocate these tasks for members with different skills. In contrast, our Manager agent can first decompose the issue into tasks and then design Developer agents to form a team. This setup improves team flexibility and adaptability, enabling the formation of teams that can meet various issues efficiently.
- **Repository Custodian**. Considering extensive files in a repository, the custodian agent's task is to locate files relevant to the issue. Unlike humans, who can browse through the entire repository, the LLM-based agent faces challenges in browsing. Although LLMs have extended context limits, their application is constrained in two aspects. First, it is a high computational cost to query each file in an entire repository for each update, particularly when some repositories update frequently. Second, the performance of LLMs degrades when the context input is long [31, 33, 67].
- **Developer**. Compared to human developers, the Developer agent can work continuously and efficiently. Therefore, scheduling the agent to work in parallel is easier than scheduling humans who require considering factors beyond the task. Additionally, although numerous developer agents are capable of generating code [23, 43], their ability to modify existing code is not equally proficient. To address this issue, our framework decomposes the code modification process into sub-operations including code generation. This approach enables Developers to leverage the benefits of automatic code generation thereby producing applicable code changes.
- **QA Engineer**. In software evolution, QA Engineers play a crucial role in maintaining software quality through code review [34, 30]. Despite their importance, code review practices are often undervalued or even overlooked [4]. Such neglect can hinder software development, illustrated by instances where developers may experience delays of up to 96 hours awaiting code review feedback [6]. To address this problem, our framework pairs each Developer agent with a QA Engineer agent, designed to offer task-specific, timely feedback. This personalized QA approach aims to boost the review process thereby better ensuring the software quality.

3.2 Collaborative Process

3.2.1 Planning

Three types of role agents engage in the planning: Repository Custodian, Manager, and Developer. This process comprises three phases: locating code files, team building, and kick-off meeting.

Locating Code Files. Firstly, the Repository Custodian employs the BM25 algorithm [45] to rank the files in the repository based on the GitHub issue description. Subsequently, the top k files are selected as potential candidates for further coding. However, as described in §2, this simple retrieval method can introduce irrelevant files, increasing the cost and reducing the effectiveness of subsequent coding process. Therefore, we filter these files based on relevance to minimize their number. While it is feasible to directly assess the relevance between each file and the issue by LLMs, queries to the LLM may contain the same code snippets as previous ones, leading to unnecessary computational costs. Considering that applying the code change often modifies a specific part of the file rather than the entire file, we propose a memory mechanism to reuse the previously queried information.

Algorithm 1 Locating.

```
Input: repository: $\mathcal{R}_i$ including files $\{f_i\}$, GitHub issue: $q_x$, LLM: $\mathcal{L}$
Config: filter top width: $k$, prompts: $\mathcal{P}$, find the latest previous version of the file and its summary: $find$
Output: candidate files: $\mathcal{C}_i^k \leftarrow \emptyset$, repository evolution memory: $\mathcal{M} \leftarrow \emptyset$
$\mathcal{R}_i \leftarrow$ BM25($\mathcal{R}_i$, $q_x$)
$\mathcal{C}_i^k \leftarrow \mathcal{R}_i[:k]$
for $f_i \in \mathcal{C}_i^k$ do
    $f_h, s_h \leftarrow find\,(f_i, \mathcal{M})$
    if $\exists\ f_h$ and len($s_h$) < len($f_i$) then
        if $h$ is $i$ then
            $s_i \leftarrow s_h$
        else
            $\Delta d \leftarrow$ diff($f_h$, $f_i$)
            $m \leftarrow \mathcal{L}(\Delta d, \mathcal{P}_1)$
            $s_i \leftarrow s_h \cup m$
        end if
    else
        $s_i \leftarrow \mathcal{L}(f_i, \mathcal{P}_2)$
    end if
    $\mathcal{M} \leftarrow \mathcal{M}$.update($\{f_i : s_i\}$)
    if $\mathcal{L}((s_i, q_x), \mathcal{P}_3)$ is false then
        $\mathcal{C}_i^k \leftarrow \mathcal{C}_i^k$ - $f_i$
    end if
end for
```

Algorithm 1 outlines the process of locating files with our designed memory $\mathcal{M}$. If a file f_i is compared for the first time with an issue q_x, the LLM $\mathcal{L}$ with prompt $\mathcal{P}_2$ compresses it into the summary s_i, where i denotes the file's version. This summary is shorter than the code content in the file and it is stored in memory for future reuse. If the file f_i has been previously compared, the latest previous version (h) of the file f_h can be found by the script $find$. Since f_i can be represented as the combination of f_h and the difference between them (Δd that be obtained via the "`git diff`" command), LLMs can understand f_i by using f_h and Δd. If the difference is small and the file f_i is long, it is valuable to reuse the previous summary s_h stored in memory rather than the content of f_i. Specifically, if the length of s_h is less than that of f_i, $\mathcal{L}$ with prompt $\mathcal{P}_1$ can summarize the code changes Δd as a "commit message" m. The combination of s_h and m forms the description of the newer version f_i, enabling the LLM $\mathcal{L}$ with prompt $\mathcal{P}_3$ to determine whether it is relevant to the issue in fewer context length. Based on their relevance, the custodian agent filters irrelevant files, allowing the Manager agent to define tasks with remaining relevant files.

Team Building. In this process, the Manager agent has the flexibility to "recruit" team members as the issue needs. Firstly, upon receiving the located files, the Manager begins with analyzing the GitHub issue for the repository and breaks them into detailed file-level tasks. Specifically, for each code file f_i in the candidate set $\mathcal{C}_i^k$, the Manager leverages the LLM $\mathcal{L}$ with the prompt $\mathcal{P}_4$ and the issue description q_x to define the corresponding file-level task t_i. One issue can be converted to multiple tasks. These tasks, along with the associated code file, are stored in a task set $\mathcal{T}_i^k$. Once a task is clarified, the Manager defines the personality role r_i of the Developer by invoking LLM $\mathcal{L}$ with the prompt $\mathcal{P}_5$ and the task t_i. By iterating through these candidate code files, the Manager agent ultimately designs a collection of Developer agent role descriptions $\mathcal{D}_i^k$, thus forming the development team. The details of the team building are shown in Algorithm 2. This approach simplifies the task for LLMs because each team member only needs to handle a sub-task rather than resolving the entire complex issue.

Algorithm 2 Making the plan.

```
Input: candidate files: $\mathcal{C}_i^k$, issue: $q_x$, LLM: $\mathcal{L}$
Config: prompts: $\mathcal{P}$
Output: tasks: $\mathcal{T}_i^k \leftarrow \emptyset$, Developer agents' role description: $\mathcal{D}_i^k \leftarrow \emptyset$, plan: $c_{main}$
for $f_i \in \mathcal{C}_i^k$ do
    $t_i \leftarrow \mathcal{L}((f_i, q_x), \mathcal{P}_4)$
    $\mathcal{T}_i^k \leftarrow \mathcal{T}_i^k \cup (f_i, t_i)$
    $r_i \leftarrow \mathcal{L}((t, q_x), \mathcal{P}_5)$
    $\mathcal{D}_i^k \leftarrow \mathcal{D}_i^k \cup r_i$
end for
recording = kick_off_meeting($\mathcal{D}_i^k$)
$\mathcal{D}_i^k \leftarrow \mathcal{L}((\mathcal{D}_i^k, recording), \mathcal{P}_6)$
$c_{main} \leftarrow \mathcal{L}(recording, \mathcal{P}_7)$
```

Kick-off Meeting. After building the team, the Manager organizes a kick-off meeting. This meeting serves two purposes: ① To confirm whether the tasks assigned by the Manager are reasonable and ensure that all Developers in the team can collaboratively resolve the issue q_x, ② To determine which Developers' tasks can be executed concurrently and which tasks have dependencies need to be sorted. The meeting takes the form of a circular speech: the Manager is responsible for opening the speech, guiding the discussion and summarizing the results, and the Developers provide their opinions based on previous discussions in turn. One example of the meeting can be found in Appendix B. After the meeting, Developers adjust their role descriptions $\mathcal{D}_i^k$ based on the discussion $recording$, and the Manager, leveraging the LLM $\mathcal{L}$ and the prompt $\mathcal{P}_7$, generates a main work plan c_{main}. This plan is presented as code, and embedded into the program for execution. The meeting makes collaboration among Developers more efficient and avoids potential conflicts.

3.2.2 Coding

Based on the empirical study on line locating and the complexity (§2), we transform the code change generation into the multi-step coding process that is designed to leverage the strengths of LLMs in code generation while mitigating their weaknesses in code change generation. Two types of agents participate in the coding process: Developers and QA Engineers. As outlined in Algorithm 3, for each task t_i and its associated code file f_i in $\mathcal{T}_i^k$, the Developer agent generates the role description of the QA Engineer a_i by the LLM $\mathcal{L}$ with the prompt $\mathcal{P}_8$. Subsequently, Developers collaborate with their QA Engineers to execute the coding tasks. During each execution of the Developer, the range of lines of code that need to be modified is firstly determined as a set of intervals $\{[s'_i, e'_i]\}$ where s'_i represents the starting line number in the i-th hunk, and e'_i is the ending line number. The determination is generated by analyzing the task content t_i and file content f_i using $\mathcal{L}$ with the prompt $\mathcal{P}_9$. These intervals split the original code file f_i into parts to be modified (*old_part*) and parts to be retained. Developers then generate new code snippets, *new_part*, by $\mathcal{L}$ with the prompt $\mathcal{P}_{10}$. The code snippets replace *old_part*, resulting in a new version of the code file f_i'. Utilizing Git tools, the code change Δd_i for this file f_i is generated. With the code change Δd_i, QA Engineer produce *review_comment* and *review_decision*, by the LLM $\mathcal{L}$ with the prompt $\mathcal{P}_{11}$. If the decision, *review_decision*, is negative (i.e., $false$), the feedback, *review_comment*, prompts Developers to revise the code in the next attempt. This iterative process continues until the code change meets the quality standards (i.e., *review_decision* is $true$) or reaches a predefined maximum number of iterations. After the iteration, the final version of the code change, Δd, is fixed, which is the ultimate modification result on each file. All generated final-version code changes during this process are merged into the repository-level code change $\mathcal{D}$ as the issue solution.

Algorithm 3 Coding task execution.

```
Input: file-task pairs set: $\mathcal{T}_i^k$, LLM: $\mathcal{L}$
Config: prompts: $\mathcal{P}$, the max of iteration: $n_{\text{max}}$
Output: code changes: $\mathcal{D}$
for $f_i, t_i \in \mathcal{T}_i^k$ do
    $a_i \leftarrow \mathcal{L}((f_i, t_i), \mathcal{P}_8)$
    for $j \in [\,0, n_{\text{max}}\,)$ do
        if $j > 0$ then
            $t_i$ = ($t_i$, review_comment)
        end if
        $\{[s_i', e_i']\} \leftarrow \mathcal{L}((f_i, t_i), \mathcal{P}_9)$
        $f_i$, old_part $\leftarrow$ split($f_i$, $\{[s_i', e_i']\}$)
        new_part $\leftarrow \mathcal{L}((f_i, t_i,$ old_part$), \mathcal{P}_{10})$
        $f_i' \leftarrow$ replace($f_i$, $\{[s_i', e_i']\}$, new_part)
        $\Delta d_i \leftarrow$ diff($f_i$, $f_i'$)
        review_comment = $\mathcal{L}((t_i, \Delta d_i), \mathcal{P}_{11})$
        review_decision = $\mathcal{L}(($review_comment$), \mathcal{P}_{11})$
        if review_decision is true then
            break
        end if
    end for
    $\Delta d \leftarrow$ diff($f_i'$, $f_i$)
    $\mathcal{D} \leftarrow \mathcal{D} \cup \Delta d$
end for
```

4 Experiments and Analysis

4.1 Setup

In the experiments, we employ the SWE-bench dataset as the evaluation benchmark because it is the latest dataset specifically designed for evaluating the performance of the GitHub issue resolution. SWE-bench comprises $2,294$ issues extracted from 12 popular Python repositories, representing real software evolution requirements. Given the observation that experimental outcomes on the 25% subset of SWE-bench align with those obtained from the entire dataset [27], we opt for the same 25% subset previously utilized in experiments for GPT-4 according to their materials [13]. Moreover, the experimental scores for the five LLMs, have been made available by them [28].

Our framework is flexible to integrate various LLMs. To compare with the scores reported by SWE-bench, GPT-4 is selected as the base LLM. Another reason for the selection is that GPT-4 shows remarkable performance on code generation and understanding as demonstrated on benchmarks such as MBPP [2] and HumanEval [12]. Claude-2 is not chosen due to the unavailability of API access.

Following SWE-bench [27], the applied and resolved ratio is used to evaluate the performance under the setting with the files requiring modification provided. The applied ratio indicates the proportion of instances where the code change is successfully generated and can be applied to the code repository by Git. The resolved ratio refers to the proportion of instances where the code change is successfully applied and passes a series of tests. Additional elaboration is provided in Appendix C.

4.2 How Effective is Our Framework? (RQ 2)

The comparative performance analysis between our framework and other LLMs on the same dataset is presented in Tab. 2. The results indicate that our framework significantly outperforms other LLMs. Notably, with a resolved ratio of 13.94%, our framework's effectiveness is eight-fold that of the base LLM, GPT-4. This substantial increase underscores our framework's capability to harness the potential of LLMs more effectively. Furthermore, when contrasted with the pre-

Table 2: The comparison of overall performance between MAGIS and baselines on SWE-bench.

Method	% Applied	% Resolved
GPT-3.5	11.67	0.84
Claude-2	49.36	4.88
GPT-4	13.24	1.74
SWE-Llama 7b	51.56	2.12
SWE-Llama 13b	49.13	4.36
MAGIS	**97.39**	**13.94**
MAGIS (w/o QA)	92.71	10.63
MAGIS (w/o hints)	94.25	10.28
MAGIS (w/o hints, w/o QA)	91.99	8.71

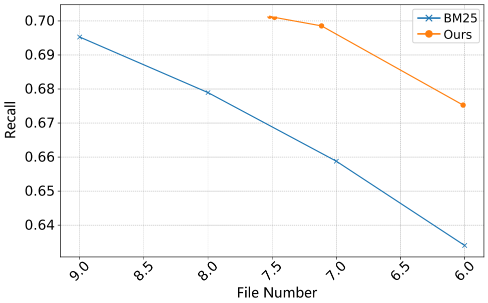


Figure 3: Comparison of recall scores between Ours and BM25.

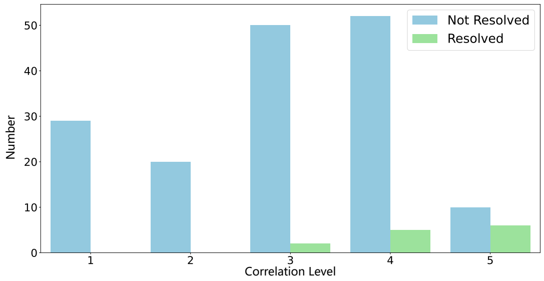


Figure 4: Distribution of the correlation score between the generated task description and the reference code change.

vious state-of-the-art LLM, Claude-2, our framework's resolved ratio exceeds that benchmark by more than two-fold. This superior performance unequivocally establishes the advance of our method.

The ablation study is designed to simulate two scenarios: ① Without QA (w/o QA): Considering the QA Engineer agent as optional within our framework, we directly evaluate the code changes generated by the Developer agent, bypassing the QA process. This scenario aims to investigate the effectiveness and necessity of QA Engineer review. ② Without hints (w/o hints): Hints refer to the textual content found in the comments section of pull requests, which are typically created before the first commit of the pull request. This setting means our framework operates without any clarifications except for the issue, despite such information being available on GitHub before the issue resolution process begins. This analysis aims to explore if the participation of humans could potentially improve the success rate of issue resolution.

Our framework shows a significant improvement in issue resolution, even without QA or hints. It achieves a resolved ratio of 8.71%, which is five times higher than that of the base LLM. This increase underscores the contribution of other agents in MAGIS to its overall performance. Furthermore, integrating cooperation with QA or hints separately can further elevate the resolved ratio by 1.92% or 1.57%, respectively. These findings underscore the value of QA Engineers and the participation of humans, as demonstrated by the resolved rates achieved through their integration.

For instance, to resolve the issue [17] from the repository `Django` [15], the developer modifies four hunks in two files [16], as shown in Fig. 15. Despite the availability of two provided files, our method opts for modifications in only one file, as illustrated in Figure 16. Remarkably, this simpler code change enables the repository to pass all requisite test cases.

Additional comparison can be found in Appendix D and E, and detailed case study is shown in Appendix H. Furthermore, the statistics on the generated code changes can be found in Appendix F.

4.3 How Effective is Our Planning Process? (RQ 3)

To investigate the effectiveness of the planning process, we analyze the Repository Custodian and Manager agent. The performance of the Repository Custodian agent is observed in the recall score versus the file number curve, as shown in Fig. 3. This curve demonstrates that our method consistently outperforms the BM25 baseline across varying numbers of selected files, indicating that our approach can identify the maximum number of relevant code files with the minimum selection.

For the Manager agent, we examined the alignment of its generated task descriptions with the reference code change by LLM. Following the study [63], we select GPT-4 as an evaluator to score the correlation between the reference code change and the generated task description. The correlation scores are determined based on a set of criteria defined in Tab. 6. A higher correlation score indicates a better alignment and thus, a more accurate and effective planning direction. The distribution of these correlation scores is presented in Fig. 4. Notably, most of the scores are 3 or above, implying that the majority of task descriptions are in the right direction concerning planning. Furthermore, the higher scores correlate with a higher probability of issue resolution, indicated by a larger proportion of "resolved" outcomes in scores 4 and 5. This signifies that when the generated task description closely aligns with the reference, there is a higher possibility of resolving the issue. The analysis

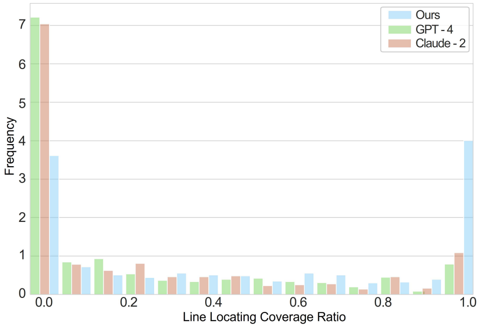


Figure 5: Comparison of line locating coverage between MAGIS (Ours) and baselines.

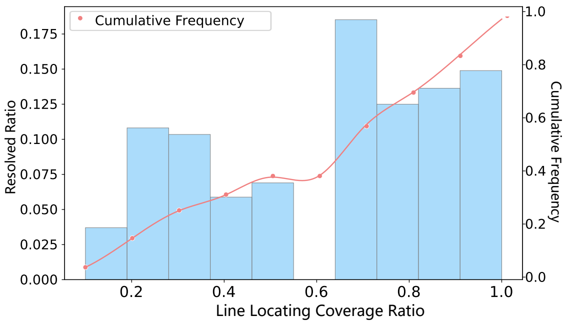


Figure 6: Resolved ratio in different line locating coverage intervals.

above demonstrates the effectiveness of both the Repository Custodian and the Manager agent in the planning process of our framework.

4.4 How Effective is Our Coding Process? (RQ 4)

To evaluate the effectiveness of the coding process in our framework, we analyze the performance of Developers in locating code lines and resolving issues of different complexity.

Fig. 5 illustrates the distribution of the line locating coverage ratio of MAGIS and the baselines. This visualization reveals that our Developer agent frequently attains a line locating coverage ratio nearing 1. Compared with baselines, the Developer agent demonstrates a pronounced preference for higher distribution values close to 1, and conversely, a reduced preference for lower distribution values near 0. Such a distribution validates the superior performance of MAGIS in locating code lines.

Further analysis is provided in Fig. 6 illustrating the relationship between the line locating coverage ratio and the issue resolved ratio within those coverages. As shown in Fig. 6, the right four bars are higher than the five left, which indicates that the resolved ratio can increase with the line locating coverage. This observation also suggests that locating lines accurately is important for issue resolution. The cumulative frequency curve, shown in orange, provides an additional analysis, indicating the cumulative proportion of issues resolved ratio up to each point along the line locating coverage. A steady increase in cumulative frequency accompanies the increase in line locating coverage, reinforcing the idea that resolving issues is more successful in areas of high coverage. The slope of the curve's left half is lower than that of the right half, indicating that the benefits of increasing the coverage ratio are less pronounced at lower coverage ratios than at higher ones. Therefore, the Developer agent should prioritize improving its capability of locating code lines.

Moreover, as shown in Tab. 3, we present a logistic regression analysis that quantifies the correlation between several complexity indices and issue resolution. The results show that GPT-4 has significant negative correlations across the number of files and functions, suggesting that as these indices increase, the likelihood of issue resolution decreases. Conversely, the negative correlations are less pronounced with our model, MAGIS, particularly in the number of files and functions, suggesting mitigation of challenges corresponding to these complexity indices.

Table 3: Correlation between the complexity indices and the issue resolution.

Method	# Files	# Functions	# Hunks	# Added LoC	# Deleted LoC	# Changed LoC
GPT-4	-25.15^*	-25.15^*	-0.06	-0.10	-0.04	-0.21
MAGIS	-1.55^*	-1.55^*	-0.12^*	-0.04^*	-0.06^*	-0.57^*

* The correlation between the index and the issue resolution is significant (P-value < 0.05).

To evaluate the performance of the QA Engineer, the ablation experiment is conducted and the results are shown in Tab. 2. As the table shows, in settings with and without hints, the presence of the QA Engineer can increase the resolved ratio by 1.57% and 3.31%, respectively. This overall enhancement substantiates the QA Engineer's contribution to improving outcomes. Furthermore, a case detailed in Appendix I underscores the QA Engineer's effectiveness.

5 Related Work

Researchers have developed LLM-based multi-agent systems, enabling more complex task completion. For instance, MetaGPT [23, 24] simulates a programming team's Standardized Operating Procedures (SOPs) and achieves leading scores on benchmarks like HumanEval [12] and MBPP [2]. Similarly, ChatDev [43] functions as a virtual development company, decomposing requirements into atomic tasks and utilizing mutual communication and self-reflection to mitigate LLM hallucinations. While these systems excel in transforming requirements into code, they often overlook the challenges of code change generation during software evolution [25]. GitHub issues include different types of requirements and most of them belong to bug fixing. Previous researchers have proposed methods to localize the bugs [65, 42] and some researchers explored various methods to automatic program repair[57, 7, 55, 3, 59, 53]. The full version of related work can be found in Appendix J.

6 Conclusion

This paper illuminates the potential of LLMs in software development, particularly in resolving GitHub issues. Our empirical study identifies the challenges of direct LLM application. To address the challenges, we propose a novel LLM-based multi-agent framework, MAGIS, enhancing issue resolution through well-designed agents' collaboration. The superiority of MAGIS on the SWE-bench against popular LLMs highlights its effectiveness, pointing towards a promising direction for integrating LLMs into software evolution workflows.

References

[1] Anthropic. Claude 2. `https://www.anthropic.com/news/claude-2`, 2023.

[2] Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. Program synthesis with large language models. *arXiv Preprint*, abs/2108.07732, 2021. URL `https://arxiv.org/abs/2108.07732`.

[3] Thomas H. Austin, Thomas Schmitz, and Cormac Flanagan. Multiple facets for dynamic information flow with exceptions. *ACM Trans. Program. Lang. Syst.*, 39(3):10:1–10:56, 2017. doi: 10.1145/3024086. URL `https://doi.org/10.1145/3024086`.

[4] Tobias Baum, Olga Liskin, Kai Niklas, and Kurt Schneider. Factors influencing code review processes in industry. In Thomas Zimmermann, Jane Cleland-Huang, and Zhendong Su, editors, *Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, FSE 2016, Seattle, WA, USA, November 13-18, 2016*, pages 85–96. ACM, 2016. doi: 10.1145/2950290.2950323. URL `https://doi.org/10.1145/2950290.2950323`.

[5] Tegawendé F. Bissyandé, David Lo, Lingxiao Jiang, Laurent Réveillère, Jacques Klein, and Yves Le Traon. Got issues? who cares about it? A large scale investigation of issue trackers from github. In *IEEE 24th International Symposium on Software Reliability Engineering, ISSRE 2013, Pasadena, CA, USA, November 4-7, 2013*, pages 188–197. IEEE Computer Society, 2013. doi: 10.1109/ISSRE.2013.6698918. URL `https://doi.org/10.1109/ISSRE.2013.6698918`.

[6] Amiangshu Bosu and Jeffrey C. Carver. Impact of developer reputation on code review outcomes in OSS projects: an empirical investigation. In Maurizio Morisio, Tore Dybå, and Marco Torchiano, editors, *2014 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '14, Torino, Italy, September 18-19, 2014*, pages 33:1–33:10. ACM, 2014. doi: 10.1145/2652524.2652544. URL `https://doi.org/10.1145/2652524.2652544`.

[7] Islem Bouzenia, Premkumar T. Devanbu, and Michael Pradel. Repairagent: An autonomous, llm-based agent for program repair. *arXiv Preprint*, abs/2403.17134, 2024. doi: 10.48550/ARXIV.2403.17134. URL `https://doi.org/10.48550/arXiv.2403.17134`.

[8] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott M. Lundberg, Harsha Nori, Hamid Palangi, Marco Túlio Ribeiro, and Yi Zhang. Sparks of artificial general intelligence: Early experiments with GPT-4. *arXiv Preprint*, abs/2303.12712, 2023. doi: 10.48550/ARXIV.2303.12712. URL `https://doi.org/10.48550/arXiv.2303.12712`.

[9] Jiayi Geng Carlos E. Jimenez, John Yang. Swe-bench lite. `https://www.swebench.com/lite.html`, 2024.

[10] Chi-Min Chan, Weize Chen, Yusheng Su, Jianxuan Yu, Wei Xue, Shanghang Zhang, Jie Fu, and Zhiyuan Liu. Chateval: Towards better llm-based evaluators through multi-agent debate. *arXiv Preprint*, abs/2308.07201, 2023. doi: 10.48550/ARXIV.2308.07201. URL `https://doi.org/10.48550/arXiv.2308.07201`.

[11] Lichang Chen, Jiuhai Chen, Heng Huang, and Minhao Cheng. PTP: boosting stability and performance of prompt tuning with perturbation-based regularizer. In Houda Bouamor, Juan Pino, and Kalika Bali, editors, *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, pages 13512–13525. Association for Computational Linguistics, 2023. URL `https://aclanthology.org/2023.emnlp-main.833`.

[12] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harrison Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. *arXiv Preprint*, abs/2107.03374, 2021. URL `https://arxiv.org/abs/2107.03374`.

[13] Google Drive. Swe-bench_api_generation. `https://drive.google.com/drive/folders/1EnrKzGAnsb_NmZKyECGmA2DrAc8ZuJ80`, 2024.

[14] Xueying Du, Mingwei Liu, Kaixin Wang, Hanlin Wang, Junwei Liu, Yixuan Chen, Jiayi Feng, Chaofeng Sha, Xin Peng, and Yiling Lou. Classeval: A manually-crafted benchmark for evaluating llms on class-level code generation, 2023.

[15] Django Software Foundation. Django. `https://github.com/django/django`, 2024.

[16] Django Software Foundation. Fixed #30255 – fixed admindocs errors when rendering docstrings without leading newlines. `https://github.com/django/django/pull/12155/files`, 2024.

[17] Django Software Foundation. #30255 (docutils reports an error rendering view docstring when the first line is not empty). `https://code.djangoproject.com/ticket/30255`, 2024.

[18] Django Software Foundation. #30664 (sqlite3 migrations can fail when used quoted db_table.). `https://code.djangoproject.com/ticket/30664`, 2024.

[19] Django Software Foundation. Custom query - django. `https://code.djangoproject.com/query`, May 11, 2024.

[20] Xinyang Geng and Hao Liu. Openllama: An open reproduction of llama, May 2023. URL `https://github.com/openlm-research/open_llama`.

[21] Inc. GitHub. Github. `https://github.com`, 2024.

[22] Inc. GitHub. Github flow. `https://docs.github.com/en/get-started/using-github/github-flow`, 2024.

[23] Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Ceyao Zhang, Jinlin Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for a multi-agent collaborative framework, 2023. URL `https://doi.org/10.48550/arXiv.2308.00352`.

[24] Sirui Hong, Yizhang Lin, Bang Liu, Bangbang Liu, Binhao Wu, Danyang Li, Jiaqi Chen, Jiayi Zhang, Jinlin Wang, Li Zhang, Lingyao Zhang, Min Yang, Mingchen Zhuge, Taicheng Guo, Tuo Zhou, Wei Tao, Wenyi Wang, Xiangru Tang, Xiangtao Lu, Xiawu Zheng, Xinbing Liang, Yaying Fei, Yuheng Cheng, Zongze Xu, and Chenglin Wu. Data interpreter: An LLM agent for data science. *arXiv Preprint*, abs/2402.18679, 2024. doi: 10.48550/ARXIV.2402.18679. URL `https://doi.org/10.48550/arXiv.2402.18679`.

[25] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John C. Grundy, and Haoyu Wang. Large language models for software engineering: A systematic literature review. *arXiv Preprint*, abs/2308.10620, 2023. doi: 10.48550/ARXIV.2308.10620. URL `https://doi.org/10.48550/arXiv.2308.10620`.

[26] Xing Hu, Xin Xia, David Lo, Zhiyuan Wan, Qiuyuan Chen, and Thomas Zimmermann. Practitioners' expectations on automated code comment generation. In *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*, pages 1693–1705. ACM, 2022. doi: 10.1145/3510003.3510152. URL `https://doi.org/10.1145/3510003.3510152`.

[27] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. URL `https://openreview.net/forum?id=VTF8yNQM66`.

[28] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Official comments to reviewer bfzn on swe-bench: Can language models resolve real-world github issues? `https://openreview.net/forum?id=VTF8yNQM66&noteId=lfJF38VxJr`, 2024.

[29] Thomas Johnsson. Attribute grammars as a functional programming paradigm. In Gilles Kahn, editor, *Functional Programming Languages and Computer Architecture, Portland, Oregon, USA, September 14-16, 1987, Proceedings*, volume 274 of *Lecture Notes in Computer Science*, pages 154–173. Springer, 1987. doi: 10.1007/3-540-18317-5_10. URL `https://doi.org/10.1007/3-540-18317-5_10`.

[30] Oleksii Kononenko, Olga Baysal, Latifa Guerrouj, Yaxin Cao, and Michael W. Godfrey. Investigating code review quality: Do people and participation matter? In Rainer Koschke, Jens Krinke, and Martin P. Robillard, editors, *2015 IEEE International Conference on Software Maintenance and Evolution, ICSME 2015, Bremen, Germany, September 29 - October 1, 2015*, pages 111–120. IEEE Computer Society, 2015. doi: 10.1109/ICSM.2015.7332457. URL `https://doi.org/10.1109/ICSM.2015.7332457`.

[31] Tianle Li, Ge Zhang, Quy Duc Do, Xiang Yue, and Wenhu Chen. Long-context llms struggle with long in-context learning. *arXiv Preprint*, abs/2404.02060, 2024. doi: 10.48550/ARXIV.2404.02060. URL `https://doi.org/10.48550/arXiv.2404.02060`.

[32] Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL `http://papers.nips.cc/paper_files/paper/2023/hash/43e9d647ccd3e4b7b5baab53f0368686-Abstract-Conference.html`.

[33] Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *arXiv Preprint*, abs/2307.03172, 2023. doi: 10.48550/ARXIV.2307.03172. URL `https://doi.org/10.48550/arXiv.2307.03172`.

[34] Shane McIntosh, Yasutaka Kamei, Bram Adams, and Ahmed E. Hassan. The impact of code review coverage and code review participation on software quality: a case study of the qt, vtk, and ITK projects. In Premkumar T. Devanbu, Sung Kim, and Martin Pinzger, editors, *11th Working Conference on Mining Software Repositories, MSR 2014, Proceedings, May 31 - June 1, 2014, Hyderabad, India*, pages 192–201. ACM, 2014. doi: 10.1145/2597073.2597076. URL `https://doi.org/10.1145/2597073.2597076`.

[35] Fangwen Mu, Xiao Chen, Lin Shi, Song Wang, and Qing Wang. Developer-intent driven code comment generation. In *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*, pages 768–780. IEEE, 2023. doi: 10.1109/ICSE48619.2023.00073. URL `https://doi.org/10.1109/ICSE48619.2023.00073`.

[36] OpenAI. GPT-4 technical report. *Arxiv Preprint*, abs/2303.08774, 2023. doi: 10.48550/ARXIV.2303.08774. URL `https://doi.org/10.48550/arXiv.2303.08774`.

[37] OpenAI. Gpt-3.5 turbo fine-tuning and api updates. `https://openai.com/blog/gpt-3-5-turbo-fine-tuning-and-api-updates`, 2023.

[38] OpenAI. Gpt-4. `https://openai.com/research/gpt-4`, 2023.

[39] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. scikitlearn. `https://github.com/scikit-learn/scikit-learn`, 2024.

[40] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. [mrg] add seeds when n_jobs=1 and use seed as random_state. `https://github.com/scikit-learn/scikit-learn/pull/9288`, 2024.

[41] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Kmeans gives slightly different result for n_jobs=1 vs. n_jobs 1. `https://github.com/scikit-learn/scikit-learn/issues/9784`, 2024.

[42] Binhang Qi, Hailong Sun, Wei Yuan, Hongyu Zhang, and Xiangxin Meng. Dreamloc: A deep relevance matching-based framework for bug localization. *IEEE Trans. Reliab.*, 71(1):235–249, 2022. doi: 10.1109/TR.2021.3104728. URL `https://doi.org/10.1109/TR.2021.3104728`.

[43] Chen Qian, Xin Cong, Wei Liu, Cheng Yang, Weize Chen, Yusheng Su, Yufan Dang, Jiahao Li, Juyuan Xu, Dahai Li, Zhiyuan Liu, and Maosong Sun. Communicative agents for software development. *arXiv Preprint*, 2023.

[44] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training, 2018.

[45] Stephen E. Robertson, Steve Walker, Susan Jones, Micheline Hancock-Beaulieu, and Mike Gatford. Okapi at TREC-3. In Donna K. Harman, editor, *Proceedings of The Third Text REtrieval Conference, TREC 1994, Gaithersburg, Maryland, USA, November 2-4, 1994*, volume 500-225 of *NIST Special Publication*, pages 109–126. National Institute of Standards and Technology (NIST), 1994. URL `http://trec.nist.gov/pubs/trec3/papers/city.ps.gz`.

[46] Jessica Shieh. Best practices for prompt engineering with openai api. *OpenAI, February https://help. openai. com/en/articles/6654000-best-practices-for-prompt-engineering-with-openai-api*, 2023.

[47] Qiushi Sun, Zhirui Chen, Fangzhi Xu, Kanzhi Cheng, Chang Ma, Zhangyue Yin, Jianing Wang, Chengcheng Han, Renyu Zhu, Shuai Yuan, Qipeng Guo, Xipeng Qiu, Pengcheng Yin, Xiaoli Li, Fei Yuan, Lingpeng Kong, Xiang Li, and Zhiyong Wu. A survey of neural code intelligence: Paradigms, advances and beyond, 2024.

[48] Yashar Talebirad and Amirhossein Nadiri. Multi-agent collaboration: Harnessing the power of intelligent LLM agents. *arXiv Preprint*, abs/2306.03314, 2023. doi: 10.48550/ARXIV.2306.03314. URL `https://doi.org/10.48550/arXiv.2306.03314`.

[49] Wei Tao, Yucheng Zhou, Yanlin Wang, Hongyu Zhang, Haofen Wang, and Wenqiang Zhang. Kadel: Knowledge-aware denoising learning for commit message generation. *ACM Trans. Softw. Eng. Methodol.*, jan 2024. ISSN 1049-331X. doi: 10.1145/3643675. URL `https://doi.org/10.1145/3643675`.

[50] The Cognition Team. Swe-bench technical report, 2024. URL `https://www.cognition-labs.com/post/swe-bench-technical-report`.

[51] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurélien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models. *arXiv Preprint*, abs/2302.13971, 2023. doi: 10.48550/ARXIV.2302.13971. URL `https://doi.org/10.48550/arXiv.2302.13971`.

[52] Michele Tufano, Anisha Agarwal, Jinu Jang, Roshanak Zilouchian Moghaddam, and Neel Sundaresan. Autodev: Automated ai-driven development, 2024. URL `https://doi.org/10.48550/arXiv.2403.08299`.

[53] Weishi Wang, Yue Wang, Shafiq Joty, and Steven C. H. Hoi. Rap-gen: Retrieval-augmented patch generation with codet5 for automatic program repair. In Satish Chandra, Kelly Blincoe, and Paolo Tonella, editors, *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2023, San Francisco, CA, USA, December 3-9, 2023*, pages 146–158. ACM, 2023. doi: 10.1145/3611643.3616256. URL `https://doi.org/10.1145/3611643.3616256`.

[54] Yue Wang, Weishi Wang, Shafiq R. Joty, and Steven C. H. Hoi. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih, editors, *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, pages 8696–8708. Association for Computational Linguistics, 2021. doi: 10.18653/v1/2021.emnlp-main.685. URL `https://doi.org/10.18653/v1/2021.emnlp-main.685`.

[55] Chu-Pan Wong, Priscila Santiesteban, Christian Kästner, and Claire Le Goues. Varfix: balancing edit expressiveness and search effectiveness in automated program repair. In Diomidis Spinellis, Georgios Gousios, Marsha Chechik, and Massimiliano Di Penta, editors, *ESEC/FSE '21: 29th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Athens, Greece, August 23-28, 2021*, pages 354–366. ACM, 2021. doi: 10.1145/3468264.3468600. URL `https://doi.org/10.1145/3468264.3468600`.

[56] Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Shaokun Zhang, Erkang Zhu, Beibin Li, Li Jiang, Xiaoyun Zhang, and Chi Wang. Autogen: Enabling next-gen LLM applications via multi-agent conversation framework. *arXiv Preprint*, abs/2308.08155, 2023. doi: 10.48550/ARXIV.2308.08155. URL `https://doi.org/10.48550/arXiv.2308.08155`.

[57] Chunqiu Steven Xia, Yuxiang Wei, and Lingming Zhang. Automated program repair in the era of large pre-trained language models. In *45th IEEE/ACM International Conference on Software Engineering, ICSE 2023, Melbourne, Australia, May 14-20, 2023*, pages 1482–1494. IEEE, 2023. doi: 10.1109/ICSE48619.2023.00129. URL `https://doi.org/10.1109/ICSE48619.2023.00129`.

[58] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent computer interfaces enable software engineering language models, 2024.

[59] He Ye and Martin Monperrus. ITER: iterative neural repair for multi-location patches. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering, ICSE 2024, Lisbon, Portugal, April 14-20, 2024*, pages 10:1–10:13. ACM, 2024. doi: 10.1145/3597503.3623337. URL `https://doi.org/10.1145/3597503.3623337`.

[60] Yuntong Zhang, Haifeng Ruan, Zhiyu Fan, and Abhik Roychoudhury. Autocoderover: Autonomous program improvement. *arXiv Preprint*, abs/2404.05427, 2024. doi: 10.48550/ARXIV.2404.05427. URL `https://doi.org/10.48550/arXiv.2404.05427`.

[61] Ziyin Zhang, Chaoyu Chen, Bingchang Liu, Cong Liao, Zi Gong, Hang Yu, Jianguo Li, and Rui Wang. Unifying the perspectives of nlp and software engineering: A survey on language models for code, 2024. URL `https://arxiv.org/abs/2311.07989`.

[62] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, Yifan Du, Chen Yang, Yushuo Chen, Zhipeng Chen, Jinhao Jiang, Ruiyang Ren, Yifan Li, Xinyu Tang, Zikang Liu, Peiyu Liu, Jian-Yun Nie, and Ji-Rong Wen. A survey of large language models. *arXiv Preprint*, abs/2303.18223, 2023. doi: 10.48550/ARXIV.2303.18223. URL `https://doi.org/10.48550/arXiv.2303.18223`.

[63] Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena. In Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine, editors, *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, 2023. URL `http://papers.nips.cc/paper_files/paper/2023/hash/91f18a1287b398d378ef22505bf41832-Abstract-Datasets_and_Benchmarks.html`.

[64] Zibin Zheng, Kaiwen Ning, Jiachi Chen, Yanlin Wang, Wenqing Chen, Lianghong Guo, and Weicheng Wang. Towards an understanding of large language models in software engineering tasks. *arXiv Preprint*, abs/2308.11396, 2023. doi: 10.48550/ARXIV.2308.11396. URL `https://doi.org/10.48550/arXiv.2308.11396`.

[65] Jian Zhou, Hongyu Zhang, and David Lo. Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports. In Martin Glinz, Gail C. Murphy, and Mauro Pezzè, editors, *34th International Conference on Software Engineering, ICSE 2012, June 2-9, 2012, Zurich, Switzerland*, pages 14–24. IEEE Computer Society, 2012. doi: 10.1109/ICSE.2012.6227210. URL `https://doi.org/10.1109/ICSE.2012.6227210`.

[66] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Wenhai Li, and Dan Ding. Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study. *IEEE Trans. Software Eng.*, 47(2):243–260, 2021. doi: 10.1109/TSE.2018.2887384. URL `https://doi.org/10.1109/TSE.2018.2887384`.

[67] Yucheng Zhou, Xiubo Geng, Tao Shen, Chongyang Tao, Guodong Long, Jian-Guang Lou, and Jianbing Shen. Thread of thought unraveling chaotic contexts. *arXiv Preprint*, abs/2311.08734, 2023. doi: 10.48550/ARXIV.2311.08734. URL `https://doi.org/10.48550/arXiv.2311.08734`.

[68] Tong Zhu, Xiaoye Qu, Daize Dong, Jiacheng Ruan, Jingqi Tong, Conghui He, and Yu Cheng. Llama-moe: Building mixture-of-experts from llama with continual pre-training. *arXiv preprint arXiv:2406.16554*, 2024. URL `https://arxiv.org/abs/2406.16554`.

A Detailed Explanation in Empirical Study

A.1 Coverage Ratio

The formula for calculating the coverage ratio is Equation 1. As it shows, for each instance of GitHub issue resolution, the range of code change (in terms of the number of lines) in the reference r is represented as a set of intervals $\boldsymbol{L}_r = \{[s_0, e_0], ..., [s_n, e_n]\}$, while the line ranges of the generated code change g is $\boldsymbol{L}_g = \{[s'_0, e'_0], ..., [s'_m, e'_m]\}$, where s and e respectively represent the starting and ending line number of each modification hunk in the file, with n hunks in the reference code change and m hunks in the generated one.

A.2 Observation on Fig. 1

As shown in Fig. 1, we observe that: ① The distribution near the coverage ratio 0 (left side of the figure) is the highest for all three LLMs, indicating that in most cases, the content generated by these models has a very low coverage ratio with the reference in terms of locating code lines. This means that these LLMs are most likely not able to accurately locate code lines that need to be modified in the process of generating the code change. ② In the distribution near the line locating coverage of 1 (right side of the figure), the three models show a consistent ranking (i.e., Claude-2 > GPT-4 > GPT-3.5) and this ranking is also consistent with the proportion of instances solved by the three models. This phenomenon suggests that the performance of LLMs in generating the code change is probably related to their ability to locate code lines accurately.

A.3 Analysis on Complexity of the Code Change

As shown in Fig. 1, compared with GPT-3.5 and GPT-4, Claude-2 exhibits a different pattern, with much lower negative correlations for the number of files and functions, which indicates that it is a more efficient approach to generate the code change for GitHub issue resolution. However, it also shows significant negative correlations across other indices such as the number of hunks, added lines of code (LoC), deleted LoC, and changed LoC.

B Kick-off Meeting Example

Figure 7 illustrates a kick-off team meeting. In this meeting, three participants are present: the Manager agent, Oliver CodeLead, and two Developer agents, Django Database Specialist and Alex Rossini. They discuss a specific issue[3], assigned tasks, and determine the workflow sequence.

C Metrics

The applied ratio indicates the proportion of instances where the code change is successfully generated and can be applied to the existing code repository using Git tools, i.e.,

$$\text{Applied Ratio} = \frac{|\mathcal{D}|}{|\mathcal{I}|}, \tag{2}$$

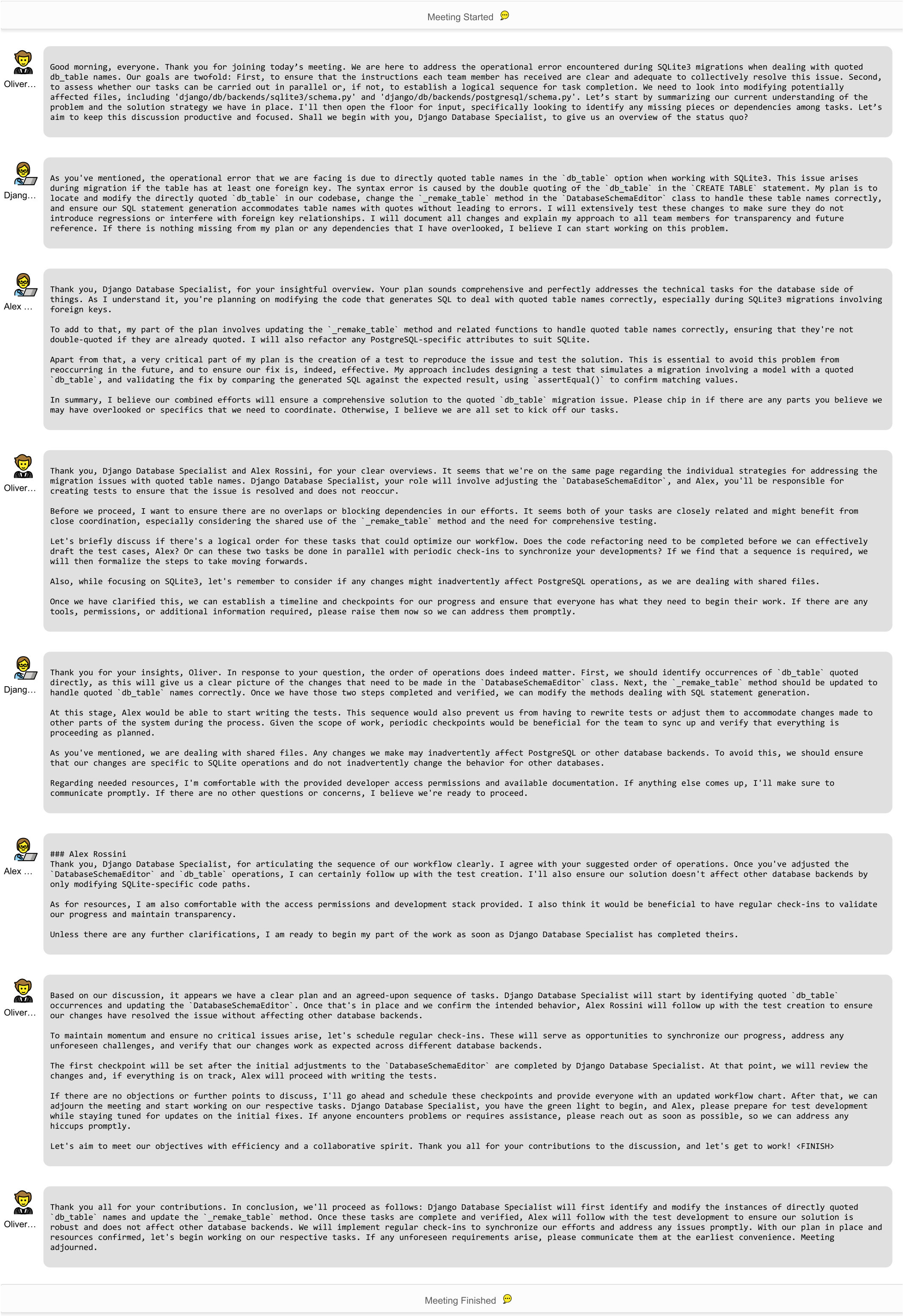
where $\mathcal{D}$ represents the set of instances in the generated code change set that could be applied to the original code repository using the "`git apply`" operation, and $\mathcal{I}$ is the set of all instances in the test set. The resolved ratio refers to the proportion of instances in which the code change is successfully applied and passed a series of tests, i.e.,

$$\text{Resolved Ratio} = \frac{\left|\sum_{i=0}^{l}(\{T_{old}(d_i)\} \cap \{T_{new}(d_i)\})\right|}{|\mathcal{I}|}, \tag{3}$$

where T_{old} denotes all the test cases that the old version of the code repository could pass, T_{new} represents all the test cases designed for new requirements, and d_i denotes the code change generated to resolve the issue in the i-th instance. Furthermore, $T(d) = \text{True}$ means that the code change d can pass all the test cases in T.

---

[3] `https://code.djangoproject.com/ticket/30664`



Meeting Started

Oliver…

Good morning, everyone. Thank you for joining today’s meeting. We are here to address the operational error encountered during SQLite3 migrations when dealing with quoted db_table names. Our goals are twofold: First, to ensure that the instructions each team member has received are clear and adequate to collectively resolve this issue. Second, to assess whether our tasks can be carried out in parallel or, if not, to establish a logical sequence for task completion. We need to look into modifying potentially affected files, including 'django/db/backends/sqlite3/schema.py' and 'django/db/backends/postgresql/schema.py'. Let’s start by summarizing our current understanding of the problem and the solution strategy we have in place. I'll then open the floor for input, specifically looking to identify any missing pieces or dependencies among tasks. Let’s aim to keep this discussion productive and focused. Shall we begin with you, Django Database Specialist, to give us an overview of the status quo?

Djang…

As you've mentioned, the operational error that we are facing is due to directly quoted table names in the `db_table` option when working with SQLite3. This issue arises during migration if the table has at least one foreign key. The syntax error is caused by the double quoting of the `db_table` in the `CREATE TABLE` statement. My plan is to locate and modify the directly quoted `db_table` in our codebase, change the `_remake_table` method in the `DatabaseSchemaEditor` class to handle these table names correctly, and ensure our SQL statement generation accommodates table names with quotes without leading to errors. I will extensively test these changes to make sure they do not introduce regressions or interfere with foreign key relationships. I will document all changes and explain my approach to all team members for transparency and future reference. If there is nothing missing from my plan or any dependencies that I have overlooked, I believe I can start working on this problem.

Alex …

Thank you, Django Database Specialist, for your insightful overview. Your plan sounds comprehensive and perfectly addresses the technical tasks for the database side of things. As I understand it, you're planning on modifying the code that generates SQL to deal with quoted table names correctly, especially during SQLite3 migrations involving foreign keys.

To add to that, my part of the plan involves updating the `_remake_table` method and related functions to handle quoted table names correctly, ensuring that they're not double-quoted if they are already quoted. I will also refactor any PostgreSQL-specific attributes to suit SQLite.

Apart from that, a very critical part of my plan is the creation of a test to reproduce the issue and test the solution. This is essential to avoid this problem from reoccurring in the future, and to ensure our fix is, indeed, effective. My approach includes designing a test that simulates a migration involving a model with a quoted `db_table`, and validating the fix by comparing the generated SQL against the expected result, using `assertEqual()` to confirm matching values.

In summary, I believe our combined efforts will ensure a comprehensive solution to the quoted `db_table` migration issue. Please chip in if there are any parts you believe we may have overlooked or specifics that we need to coordinate. Otherwise, I believe we are all set to kick off our tasks.

Oliver…

Thank you, Django Database Specialist and Alex Rossini, for your clear overviews. It seems that we're on the same page regarding the individual strategies for addressing the migration issues with quoted table names. Django Database Specialist, your role will involve adjusting the `DatabaseSchemaEditor`, and Alex, you'll be responsible for creating tests to ensure that the issue is resolved and does not reoccur.

Before we proceed, I want to ensure there are no overlaps or blocking dependencies in our efforts. It seems both of your tasks are closely related and might benefit from close coordination, especially considering the shared use of the `_remake_table` method and the need for comprehensive testing.

Let's briefly discuss if there's a logical order for these tasks that could optimize our workflow. Does the code refactoring need to be completed before we can effectively draft the test cases, Alex? Or can these two tasks be done in parallel with periodic check-ins to synchronize your developments? If we find that a sequence is required, we will then formalize the steps to take moving forwards.

Also, while focusing on SQLite3, let's remember to consider if any changes might inadvertently affect PostgreSQL operations, as we are dealing with shared files.

Once we have clarified this, we can establish a timeline and checkpoints for our progress and ensure that everyone has what they need to begin their work. If there are any tools, permissions, or additional information required, please raise them now so we can address them promptly.

Djang…

Thank you for your insights, Oliver. In response to your question, the order of operations does indeed matter. First, we should identify occurrences of `db_table` quoted directly, as this will give us a clear picture of the changes that need to be made in the `DatabaseSchemaEditor` class. Next, the `_remake_table` method should be updated to handle quoted `db_table` names correctly. Once we have those two steps completed and verified, we can modify the methods dealing with SQL statement generation.

At this stage, Alex would be able to start writing the tests. This sequence would also prevent us from having to rewrite tests or adjust them to accommodate changes made to other parts of the system during the process. Given the scope of work, periodic checkpoints would be beneficial for the team to sync up and verify that everything is proceeding as planned.

As you've mentioned, we are dealing with shared files. Any changes we make may inadvertently affect PostgreSQL or other database backends. To avoid this, we should ensure that our changes are specific to SQLite operations and do not inadvertently change the behavior for other databases.

Regarding needed resources, I'm comfortable with the provided developer access permissions and available documentation. If anything else comes up, I'll make sure to communicate promptly. If there are no other questions or concerns, I believe we're ready to proceed.

Alex …

Alex Rossini
Thank you, Django Database Specialist, for articulating the sequence of our workflow clearly. I agree with your suggested order of operations. Once you've adjusted the `DatabaseSchemaEditor` and `db_table` operations, I can certainly follow up with the test creation. I'll also ensure our solution doesn't affect other database backends by only modifying SQLite-specific code paths.

As for resources, I am also comfortable with the access permissions and development stack provided. I also think it would be beneficial to have regular check-ins to validate our progress and maintain transparency.

Unless there are any further clarifications, I am ready to begin my part of the work as soon as Django Database Specialist has completed theirs.

Oliver…

Based on our discussion, it appears we have a clear plan and an agreed-upon sequence of tasks. Django Database Specialist will start by identifying quoted `db_table` occurrences and updating the `DatabaseSchemaEditor`. Once that's in place and we confirm the intended behavior, Alex Rossini will follow up with the test creation to ensure our changes have resolved the issue without affecting other database backends.

To maintain momentum and ensure no critical issues arise, let's schedule regular check-ins. These will serve as opportunities to synchronize our progress, address any unforeseen challenges, and verify that our changes work as expected across different database backends.

The first checkpoint will be set after the initial adjustments to the `DatabaseSchemaEditor` are completed by Django Database Specialist. At that point, we will review the changes and, if everything is on track, Alex will proceed with writing the tests.

If there are no objections or further points to discuss, I'll go ahead and schedule these checkpoints and provide everyone with an updated workflow chart. After that, we can adjourn the meeting and start working on our respective tasks. Django Database Specialist, you have the green light to begin, and Alex, please prepare for test development while staying tuned for updates on the initial fixes. If anyone encounters problems or requires assistance, please reach out as soon as possible, so we can address any hiccups promptly.

Let's aim to meet our objectives with efficiency and a collaborative spirit. Thank you all for your contributions to the discussion, and let's get to work! <FINISH>

Oliver…

Thank you all for your contributions. In conclusion, we'll proceed as follows: Django Database Specialist will first identify and modify the instances of directly quoted `db_table` names and update the `_remake_table` method. Once these tasks are complete and verified, Alex will follow with the test development to ensure our solution is robust and does not affect other database backends. We will implement regular check-ins to synchronize our efforts and address any issues promptly. With our plan in place and resources confirmed, let's begin working on our respective tasks. If any unforeseen requirements arise, please communicate them at the earliest convenience. Meeting adjourned.

Meeting Finished

Figure 7: Kick-off meeting to resolve the issue [18].

The recall score versus file number curve is used to measure the effectiveness of locating code files to be modified. The recall score refers to the proportion of files that are successfully located out of all the files that require modification. The formula for calculating the file locating recall score for the i-th instance is as follows:

$$\text{Recall} = \frac{|\mathcal{G}_i \cap \mathcal{R}_i|}{|\mathcal{R}_i|} \times 100\%, \tag{4}$$

where $\mathcal{G}_i = \sum_{j=0}^{n} g_{i,j}$ represents the set of file paths located by our framework, with each file path in the set denoted as $g_{i,j}$ and the total number of files as n; $\mathcal{R}_i = \sum_{k=0}^{m} r_{i,k}$ denotes the paths of the files that need to be modified, with each reference file path denoted as $r_{i,k}$ and the total file number as m. In this curve, "file number" refers to the average number of files that need to be processed across all instances to achieve the given recall score. Specifically, it illustrates how many files averagely need to be located by our framework before reaching the recall score denoted by the curve at any point. This metric represents both the effectiveness and efficiency of file locating.

D Comparison Result on SWE-bench Lite

Recently, some contemporaneous works, e.g., AutoCodeRover [60] and SWE-Agent [58], have been proposed for this task. These methods are evaluated using SWE-bench lite, a canonical subset of SWE-bench, which is recommended for evaluation [9]. Considering budget constraints, we conducted experiments on SWE-bench lite to compare with them on the same issues' resolution.

The experimental results are shown in Tab. 4. MAGIS achieves the highest resolved ratio, 25.33%, than other baselines. The performance of MAGIS slightly decreased when evaluated without QA, reaching 23.33%, and dropped under the other two ablation settings. This comparative study underscores the robustness of MAGIS, particularly when provided with comprehensive inputs, and highlights the impact of QA and hints on its performance. The results indicate that while new methods like AutoCodeRover and SWE-Agent show promise, MAGIS remains an effective method for GitHub issue resolution.

Table 4: The comparison of overall performance between MAGIS and baselines on SWE-bench lite.

Method	AutoCodeRover	SWE-Agent	MAGIS			
			Full	w/o QA	w/o hints	w/o (hints, QA)
Resolved	16.11% (22.33%*)	18.00%	**25.33%**	23.33%	16.67%	16.00%

* Note that 16.11 is the average scores among 3 runs while 22.33 is under the union of from the 3 runs.

E Comparison with Devin

Devin is a novel agent for software development [50], and its performance has also been assessed using the SWE-bench. However, the evaluation dataset employed by Devin differs from the subset used for experiments with GPT-4 reported by the paper of SWE-bench [27]. An analysis of the repository name and pull request ID of each instance reveals that only 140 instances coverage between the two datasets.

Within the shared pool of 140 instances, our framework successfully resolves 21 (15%) issues, surpassing Devin's resolution of 18 (12.86%) issues [4]. This comparison, however, may not be entirely equitable. Devin's possible underlying LLM is unknown, and it possesses the capability to integrate feedback from the environment. Moreover, Devin's reported scores are under the setting given the entire repository, and it operates with "common developer tools including the shell, code editor, and browser", and "agents with internet access could potentially find external information through other methods" as detailed at the report [5]. In contrast, our approach solely relies on the shell, without the need of any additional external tools.

---

[4] `https://github.com/CognitionAI/devin-swebench-results/tree/main/output_diffs/pass`

[5] `https://www.cognition-labs.com/introducing-devin`

For running time, 72% of instances resolved by Devin require greater than 10 minutes to complete. In contrast, our framework finalizes each resolved issue within approximately 3 minutes. On average, our framework completes the processing of each instance in under 5 minutes, demonstrating its capability to assist in resolving GitHub issues with minimal time expenditure.

F Statistics on the Generated Code Changes

This section provides statistics on code changes corresponding to resolved issues and those applicable but unresolved using our framework.

The statistics on the code change for instances with resolved issues are presented in Tab. 5. Overall, the statistical information of the generated code changes for these instances, such as the average number of code files, functions, hunks, and deleted lines, all differ slightly (not exceeding 0.3) from the reference solutions written by humans. This indicates that for these instances, the complexity of the code change generated by our framework is similar to that of humans. Furthermore, the maximum values observed in the table reveal that our framework can implement code modifications involving two files, four hunks, and $1,655$ lines modification, with single modifications reaching up to 190 lines. Results demonstrate the effectiveness of our method in resolving complex issues that need to modify the code file on multiple locations and with long context.

Specifically, the distribution of the number of modified lines for the resolving instances is shown in Fig. 8. We observe that the distribution of the number of modified lines in our framework for the solved instances exceeds that of the reference solution, especially in terms of the number of added lines being significantly higher than the reference. Upon manual inspection, we found that the generation results provided by our framework often contained more comment information, which led to an increase in the total number of modified lines. For example, Fig. 10 displays the generation result of our framework. Lines $365, 368, 371, 374, 383$ in the new version file correspond to the comment for the added code. These natural language descriptions are valuable in actual software evolution [26, 35]. In contrast, Fig. 12 shows a human-written solution lacking such explanatory comments, which might disadvantage software maintainers in reading and understanding.

The statistics on the code change for instances without resolved issues are shown in Tab. 5. From the table, our framework can generate applicable code changes including up to 13 files and 28 hunks, and the location of the modifications can be as far as line $7,150$, with a single modification reaching up to $9,367$ lines. These results suggest that our method has a strong adaptability in generating applicable code changes. However, considering that these code changes have not passed all the potential test cases they could pass, which indicates that there is still room for improvement.

To further analyze the reasons behind the failure of test cases in these instances, we have quantified the distribution of the lengths of code changes in the unresolved instances, as shown in Fig. 9. From the figure, we observe that for unresolved instances, the framework tends to delete a larger number of lines while adding fewer lines, in contrast to the distribution of human-written changes. This discrepancy may point to different repair strategies or attitudes towards problem-solving, where the framework presented herein might prefer to reduce errors by removing potentially problematic code, whereas human developers may lean towards adding new code to address issues.

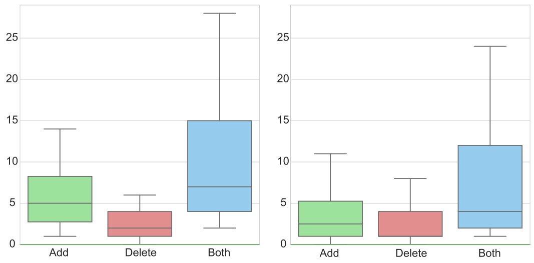


Figure 8: Distribution of the LoC in the resolved instances.

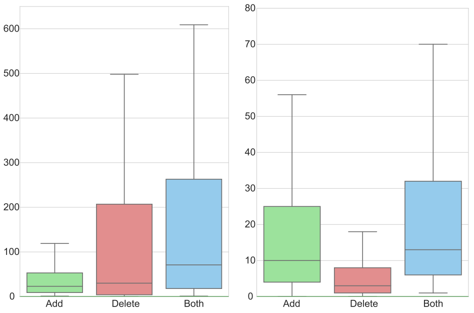
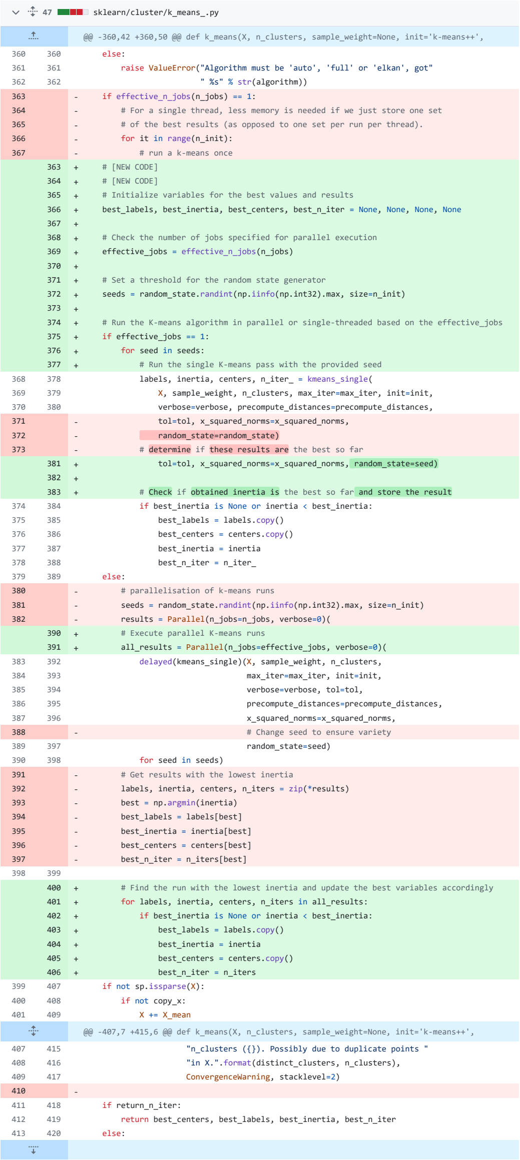
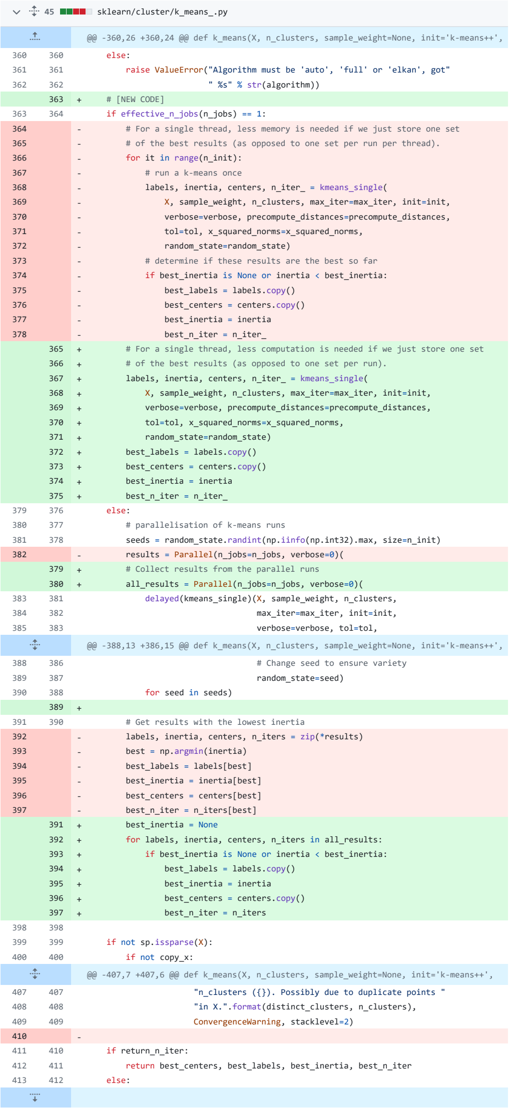


Figure 9: Distribution of the LoC in the applied but not resolved instances.



```diff
47 sklearn/cluster/k_means_.py
@@ -360,42 +360,50 @@ def k_means(X, n_clusters, sample_weight=None, init='k-means++',
     else:
         raise ValueError("Algorithm must be 'auto', 'full' or 'elkan', got"
                          " %s" % str(algorithm))
-    if effective_n_jobs(n_jobs) == 1:
-        # For a single thread, less memory is needed if we just store one set
-        # of the best results (as opposed to one set per run per thread).
-        for it in range(n_init):
-            # run a k-means once
+    # [NEW CODE]
+    # [NEW CODE]
+    # Initialize variables for the best values and results
+    best_labels, best_inertia, best_centers, best_n_iter = None, None, None, None
+
+    # Check the number of jobs specified for parallel execution
+    effective_jobs = effective_n_jobs(n_jobs)
+
+    # Set a threshold for the random state generator
+    seeds = random_state.randint(np.iinfo(np.int32).max, size=n_init)
+
+    # Run the K-means algorithm in parallel or single-threaded based on the effective_jobs
+    if effective_jobs == 1:
+        for seed in seeds:
+            # Run the single K-means pass with the provided seed
             labels, inertia, centers, n_iter_ = kmeans_single(
                 X, sample_weight, n_clusters, max_iter=max_iter, init=init,
                 verbose=verbose, precompute_distances=precompute_distances,
-                tol=tol, x_squared_norms=x_squared_norms,
-                random_state=random_state)
-            # determine if these results are the best so far
+                tol=tol, x_squared_norms=x_squared_norms, random_state=seed)
+
+            # Check if obtained inertia is the best so far and store the result
             if best_inertia is None or inertia < best_inertia:
                 best_labels = labels.copy()
                 best_centers = centers.copy()
                 best_inertia = inertia
                 best_n_iter = n_iter_
     else:
-        # parallelisation of k-means runs
-        seeds = random_state.randint(np.iinfo(np.int32).max, size=n_init)
-        results = Parallel(n_jobs=n_jobs, verbose=0)(
+        # Execute parallel K-means runs
+        all_results = Parallel(n_jobs=effective_jobs, verbose=0)(
             delayed(kmeans_single)(X, sample_weight, n_clusters,
                                    max_iter=max_iter, init=init,
                                    verbose=verbose, tol=tol,
                                    precompute_distances=precompute_distances,
                                    x_squared_norms=x_squared_norms,
-                                   # Change seed to ensure variety
                                    random_state=seed)
             for seed in seeds)
-        # Get results with the lowest inertia
-        labels, inertia, centers, n_iters = zip(*results)
-        best = np.argmin(inertia)
-        best_labels = labels[best]
-        best_inertia = inertia[best]
-        best_centers = centers[best]
-        best_n_iter = n_iters[best]
 
+        # Find the run with the lowest inertia and update the best variables accordingly
+        for labels, inertia, centers, n_iters in all_results:
+            if best_inertia is None or inertia < best_inertia:
+                best_labels = labels.copy()
+                best_inertia = inertia
+                best_centers = centers.copy()
+                best_n_iter = n_iters
     if not sp.issparse(X):
         if not copy_x:
             X += X_mean
@@ -407,7 +415,6 @@ def k_means(X, n_clusters, sample_weight=None, init='k-means++',
                       "n_clusters ({}). Possibly due to duplicate points "
                       "in X.".format(distinct_clusters, n_clusters),
                       ConvergenceWarning, stacklevel=2)
-
     if return_n_iter:
         return best_centers, best_labels, best_inertia, best_n_iter
     else:
```

Figure 10: Case from `scikit-learn` (ours, after review) for the issue [41].

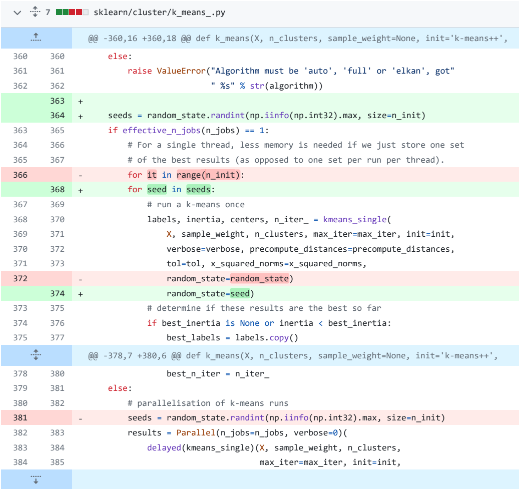


```diff
45 sklearn/cluster/k_means_.py
@@ -360,26 +360,24 @@ def k_means(X, n_clusters, sample_weight=None, init='k-means++',
     else:
         raise ValueError("Algorithm must be 'auto', 'full' or 'elkan', got"
                          " %s" % str(algorithm))
+    # [NEW CODE]
     if effective_n_jobs(n_jobs) == 1:
-        # For a single thread, less memory is needed if we just store one set
-        # of the best results (as opposed to one set per run per thread).
-        for it in range(n_init):
-            # run a k-means once
-            labels, inertia, centers, n_iter_ = kmeans_single(
-                X, sample_weight, n_clusters, max_iter=max_iter, init=init,
-                verbose=verbose, precompute_distances=precompute_distances,
-                tol=tol, x_squared_norms=x_squared_norms,
-                random_state=random_state)
-            # determine if these results are the best so far
-            if best_inertia is None or inertia < best_inertia:
-                best_labels = labels.copy()
-                best_centers = centers.copy()
-                best_inertia = inertia
-                best_n_iter = n_iter_
+        # For a single thread, less computation is needed if we just store one set
+        # of the best results (as opposed to one set per run).
+        labels, inertia, centers, n_iter_ = kmeans_single(
+            X, sample_weight, n_clusters, max_iter=max_iter, init=init,
+            verbose=verbose, precompute_distances=precompute_distances,
+            tol=tol, x_squared_norms=x_squared_norms,
+            random_state=random_state)
+        best_labels = labels.copy()
+        best_centers = centers.copy()
+        best_inertia = inertia
+        best_n_iter = n_iter_
     else:
         # parallelisation of k-means runs
         seeds = random_state.randint(np.iinfo(np.int32).max, size=n_init)
-        results = Parallel(n_jobs=n_jobs, verbose=0)(
+        # Collect results from the parallel runs
+        all_results = Parallel(n_jobs=n_jobs, verbose=0)(
             delayed(kmeans_single)(X, sample_weight, n_clusters,
                                    max_iter=max_iter, init=init,
                                    verbose=verbose, tol=tol,
@@ -388,13 +386,15 @@ def k_means(X, n_clusters, sample_weight=None, init='k-means++',
                                    # Change seed to ensure variety
                                    random_state=seed)
             for seed in seeds)
+
         # Get results with the lowest inertia
-        labels, inertia, centers, n_iters = zip(*results)
-        best = np.argmin(inertia)
-        best_labels = labels[best]
-        best_inertia = inertia[best]
-        best_centers = centers[best]
-        best_n_iter = n_iters[best]
+        best_inertia = None
+        for labels, inertia, centers, n_iters in all_results:
+            if best_inertia is None or inertia < best_inertia:
+                best_labels = labels.copy()
+                best_inertia = inertia
+                best_centers = centers.copy()
+                best_n_iter = n_iters
 
     if not sp.issparse(X):
         if not copy_x:
@@ -407,7 +407,6 @@ def k_means(X, n_clusters, sample_weight=None, init='k-means++',
                       "n_clusters ({}). Possibly due to duplicate points "
                       "in X.".format(distinct_clusters, n_clusters),
                       ConvergenceWarning, stacklevel=2)
-
     if return_n_iter:
         return best_centers, best_labels, best_inertia, best_n_iter
     else:
```

Figure 11: Case from `scikit-learn` (ours, before review) for the issue [41].

Moreover, a comparison between the resolved instances and not resolved ones shown in Tab. 5 reveals that the latter contains a higher overall number of files, hunks, and changed lines of code. These instances, involving more modification locations, correspond to more complex scenarios. This phenomenon suggests that the performance of our framework in resolving such complex issues requires further enhancement.

Furthermore, the variability in difficulty across different software repositories may influence the effectiveness of code changes. To this end, we compile statistics on the resolved ratios in various software repositories, as shown in Fig. 13. From the figure, we observe that there is a significant variation in the resolved ratios across different repositories in our framework. Some repositories have a resolved ratio as high as 40%, while others are close to 0%. This suggests that the differences among various software such as code structure and coding style can impact the generation and application of the code change.



```
7 sklearn/cluster/k_means_.py
@@ -360,16 +360,18 @@ def k_means(X, n_clusters, sample_weight=None, init='k-means++',
        else:
            raise ValueError("Algorithm must be 'auto', 'full' or 'elkan', got"
                             " %s" % str(algorithm))
+
+       seeds = random_state.randint(np.iinfo(np.int32).max, size=n_init)
        if effective_n_jobs(n_jobs) == 1:
            # For a single thread, less memory is needed if we just store one set
            # of the best results (as opposed to one set per run per thread).
-           for it in range(n_init):
+           for seed in seeds:
                # run a k-means once
                labels, inertia, centers, n_iter_ = kmeans_single(
                    X, sample_weight, n_clusters, max_iter=max_iter, init=init,
                    verbose=verbose, precompute_distances=precompute_distances,
                    tol=tol, x_squared_norms=x_squared_norms,
-                   random_state=random_state)
+                   random_state=seed)
                # determine if these results are the best so far
                if best_inertia is None or inertia < best_inertia:
                    best_labels = labels.copy()
@@ -378,7 +380,6 @@ def k_means(X, n_clusters, sample_weight=None, init='k-means++',
                    best_n_iter = n_iter_
        else:
            # parallelisation of k-means runs
-           seeds = random_state.randint(np.iinfo(np.int32).max, size=n_init)
            results = Parallel(n_jobs=n_jobs, verbose=0)(
                delayed(kmeans_single)(X, sample_weight, n_clusters,
                                       max_iter=max_iter, init=init,
```

Figure 12: Case from `scikit-learn` (gold) [40].

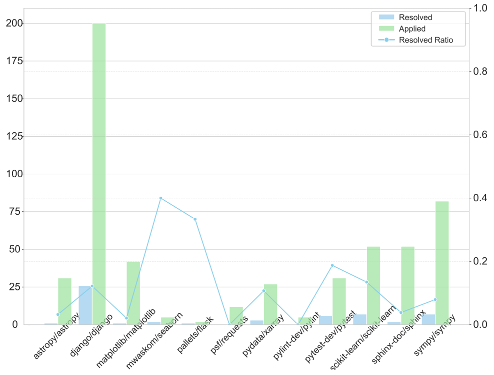


Figure 13: The number of applied and resolved instances in different repositories.

Table 5: The statistical analysis of our framework on resolved and applied but not resolved instances.

	Resolved Instances						Applied but Not Resolved Instances					
	MAGIS			Gold			MAGIS			Gold		
	Min	Max	Avg.	Min	Max	Avg.	Min	Max	Avg.	Min	Max	Avg.
# Code Files	1	2	1.02	1	2	1.04	1	13	1.50	1	18	1.61
# Functions	1	2	1.02	1	2	1.04	1	13	1.50	1	18	1.61
# Hunks	1	4	1.45	1	6	1.66	1	28	2.52	1	52	3.72
# Added Lines	1	146	9.75	0	38	4.34	1	920	40.38	0	3,050	28.27
# Deleted Lines	0	77	5.27	0	115	5.16	0	9,160	327.27	0	2,975	14.51
Change Start Index	1	1,655	270.12	1	1,657	256.09	1	4,568	424.84	0	6,651	485.01
Change End Index	22	1,665	301.68	0	1,666	315.05	9	7,150	513.13	0	6,658	728.96
# Changed Lines	2	190	15.02	1	115	9.50	1	9,367	367.65	1	6,025	42.79

G Evaluation on Task Description

Since there is no ground truth for the task descriptions generated by the Manager, we utilize GPT-4 to simulate human evaluation and score each task description based on its corresponding reference code change. Table 6 illustrates the standards used by GPT-4 to assess the correlation between the task description and the code change. The score given by GPT-4 is considered the performance metric for the task description.

Table 6: The meaning of scores in GPT-4 evaluation on the correlation between the generated task description and the reference code change.

Score	Meaning
1	The code changes are unrelated to the task description.
2	The code changes address a minor part of the task but are largely irrelevant.
3	The code changes partially meet the task requirements but lack completeness or accuracy.
4	The code changes are relevant and mostly complete, with minor discrepancies from the task description.
5	The code changes perfectly align with the task description, fully addressing all specified requirements with high accuracy and completeness.

H Case Study

Fig. 14 illustrates the detailed process of our framework used to resolve the issue from the `Django` repository [15] as described in the following ticket [6]. To address this issue, two candidate files were identified for modification. Based on the issue description and the candidate files, the Manager defined two file-level tasks. For these tasks, two Developers were assigned: Django Database Specialist (Developer I) and Alex Rossini (Developer II). Following a kick-off meeting attended by both Developers and Managers, the Django Database Specialist commenced work first, followed by Alex Rossini. During the coding phase, Developer I identified the code lines to be modified and generated the new code to replace them. The initial code changes made by Developer I were approved by the QA Engineer. Developer II made three attempts during the coding process. The QA Engineer provided review comments on the first two attempts. Ultimately, both Developers completed their coding tasks, and the merged results from their code changes passed all necessary tests.

Fig. 15 shows a reference issue resolution result, which resolves the issue [7] from the repository `Django` [15], the human developer modifies four hunks in two files [16]. Despite the presence of modifications in two files, our method focuses on changes in only one file, as shown in Figure 16. Notably, this simpler modification allows the repository to pass all necessary test cases.

I The performance of the QA Engineer Agent

Fig. 12 shows an issue [41] from the repository `scikit-learn` [39] and the reference code change [40]. During the flow of our framework, the Developer firstly modifies the code as shown in Fig. 11 but the parameter`random_state` (Line 371 in the new-version code) of the function `kmeans_single` is not assigned the right number in `seeds`. After the erroneous modification was made, the QA Engineer identified the mistake and provided feedback. Their commentary highlighted the issue: “This code change modifies the implementation of K-means algorithm and doesn’t seem entirely correct”. They further elaborated, “Running the algorithm just one time could lead to worse results, compared to running it multiple times (n_init times) and choosing the best result, as was originally done”. This critique specifically targets the flaw associated with the iterative process (“running times”). With the help of the QA Engineer, the Developer further revise the code, and the final code change is shown in Fig. 10. All of the necessary test cases are passed after applying this code change.

J Related Work (Detailed)

J.1 Large Language Models

Large Language Models (LLMs) refer to the pre-trained language models that contain a large number of parameters [62]. The parameter counts of these models typically range in the tens or hundreds of billions. Popular LLMs include the Generative Pre-trained Transformer (GPT) series, such as GPT-3 [44], GPT-4 [38], and the open-source LLaMA [51] which publicly shares its weight information. The first version of the open-source model LLaMA has parameters ranging from 7 billion to 65 billion. Many researchers [68, 20] have built upon the foundation of LLaMA, implementing enhancements to forge new LLMs. These LLMs have demonstrated formidable natural language generation capabilities in general scenarios, with GPT-4, in particular, standing out [32, 63]. It has consistently maintained the top position in several rankings, including code generation, reflecting its significant potential in tasks related to software engineering [25].

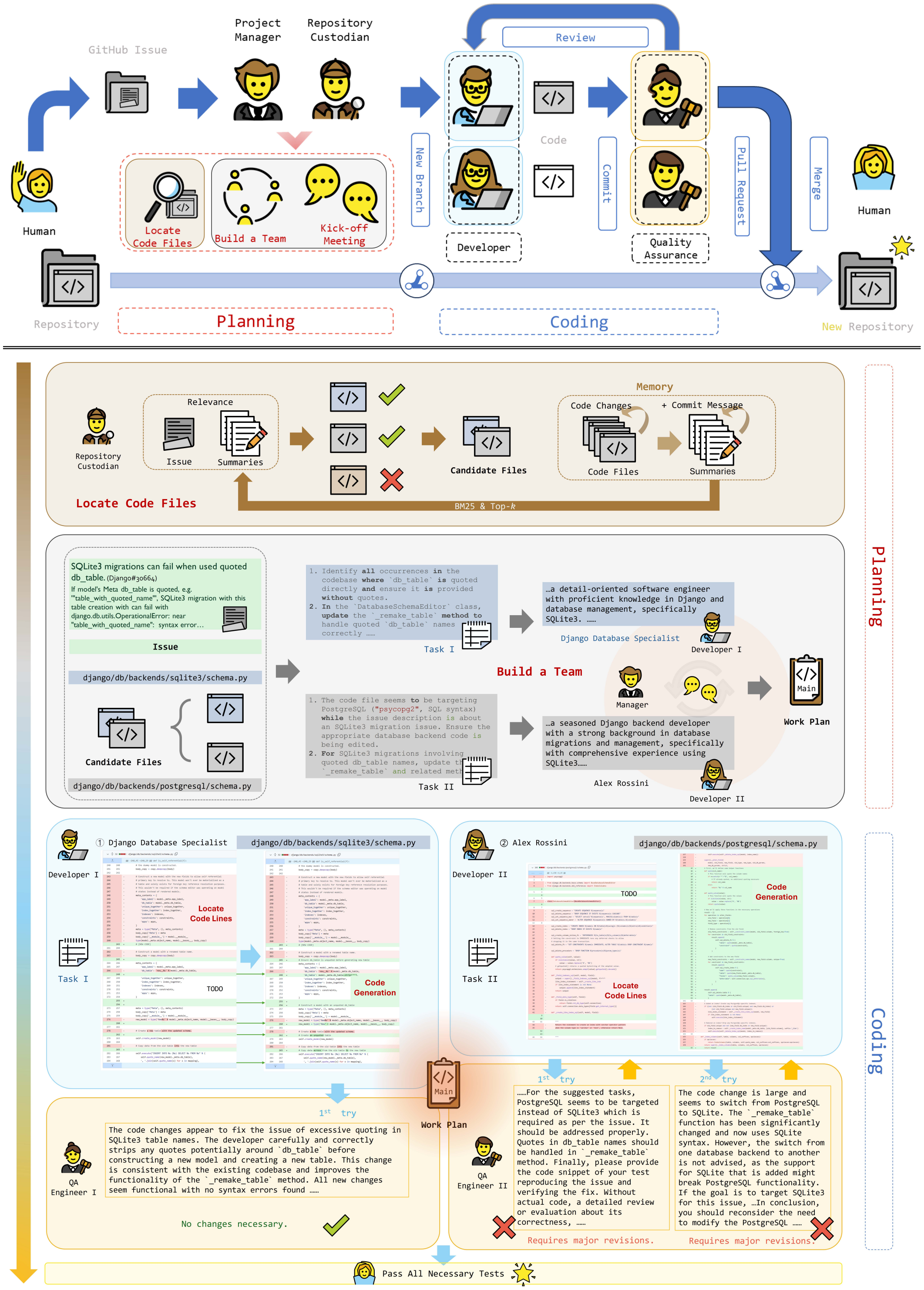
J.2 LLM-Based Multi-Agent System

With the powerful text generation capabilities of LLMs, many researchers [23, 48, 10, 56, 43, 52, 61] have explored the construction of LLM-based Multi-Agent Systems, enabling them to accomplish tasks beyond the capabilities of the LLMs themselves. For example, MetaGPT [23], which simulates the Standardized Operating Procedures (SOPs) of a programming team, completing tasks including definition, design, planning, coding, and testing through constructed roles (e.g.,

---

[6] `https://code.djangoproject.com/ticket/30664`

[7] `https://code.djangoproject.com/ticket/30255`



SQLite3 migrations can fail when used quoted db_table. (Django#30664)

If model's Meta db_table is quoted, e.g. '"table_with_quoted_name"', SQLite3 migration with this table creation with can fail with django.db.utils.OperationalError: near "table_with_quoted_name": syntax error…

1. Identify **all** occurrences **in** the codebase **where** \`db_table\` **is** quoted directly **and** ensure it **is** provided **without** quotes.
2. **In** the \`DatabaseSchemaEditor\` class, **update** the \`_remake_table\` **method** **to** handle quoted \`db_table\` names correctly ……

1. The code file seems **to** be targeting PostgreSQL ("psycopg2", SQL syntax) **while** the issue description is about an SQLite3 migration issue. Ensure the appropriate database backend code is being edited.
2. **For** SQLite3 migrations involving quoted db_table names, update th \`_remake_table\` and related meth

…a detail-oriented software engineer with proficient knowledge in Django and database management, specifically SQLite3. ……

…a seasoned Django backend developer with a strong background in database migrations and management, specifically with comprehensive experience using SQLite3……

The code changes appear to fix the issue of excessive quoting in SQLite3 table names. The developer carefully and correctly strips any quotes potentially around \`db_table\` before constructing a new model and creating a new table. This change is consistent with the existing codebase and improves the functionality of the \`_remake_table\` method. All new changes seem functional with no syntax errors found ……

No changes necessary. ✓

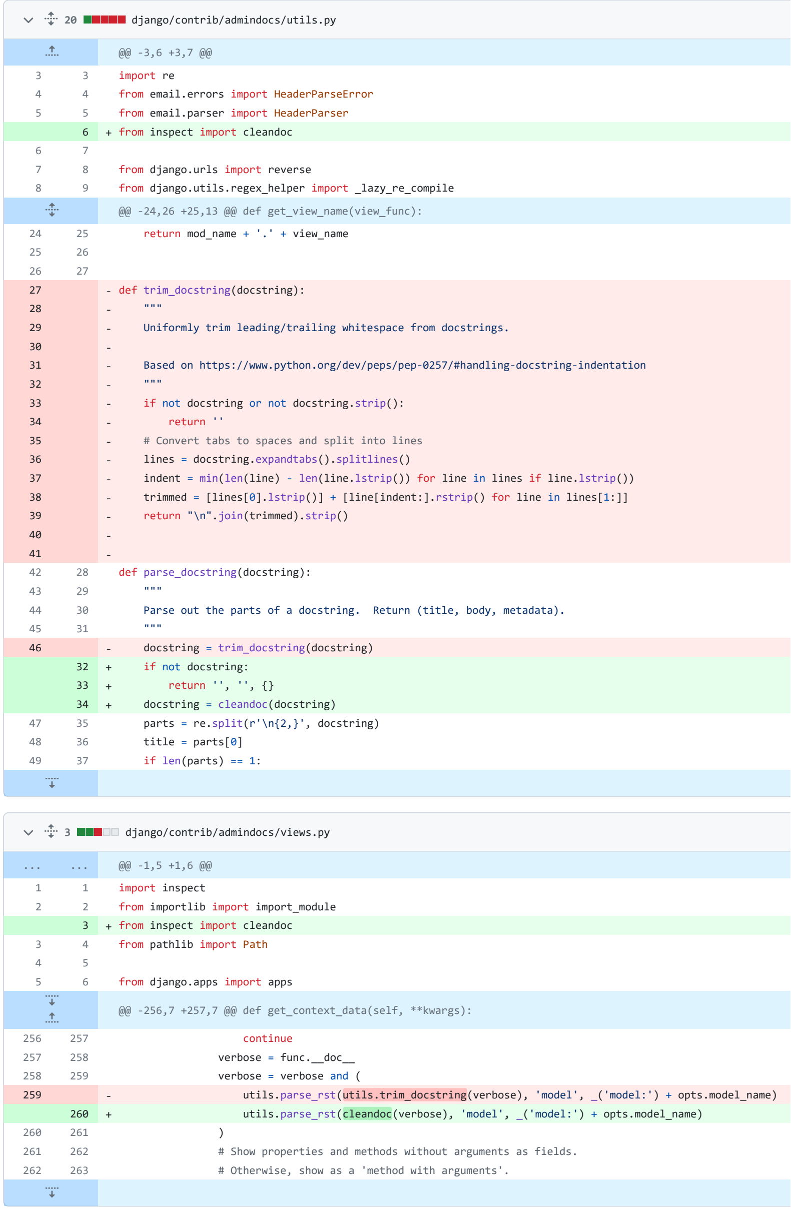
……For the suggested tasks, PostgreSQL seems to be targeted instead of SQLite3 which is required as per the issue. It should be addressed properly. Quotes in db_table names should be handled in \`_remake_table\` method. Finally, please provide the code snippet of your test reproducing the issue and verifying the fix. Without actual code, a detailed review or evaluation about its correctness, ……

Requires major revisions.

The code change is large and seems to switch from PostgreSQL to SQLite. The \`_remake_table\` function has been significantly changed and now uses SQLite syntax. However, the switch from one database backend to another is not advised, as the support for SQLite that is added might break PostgreSQL functionality. If the goal is to target SQLite3 for this issue, …In conclusion, you should reconsider the need to modify the PostgreSQL ……

Requires major revisions.

Figure 14: Detailed overview of our framework, MAGIS (Kick-off meeting refers to Fig. 7).



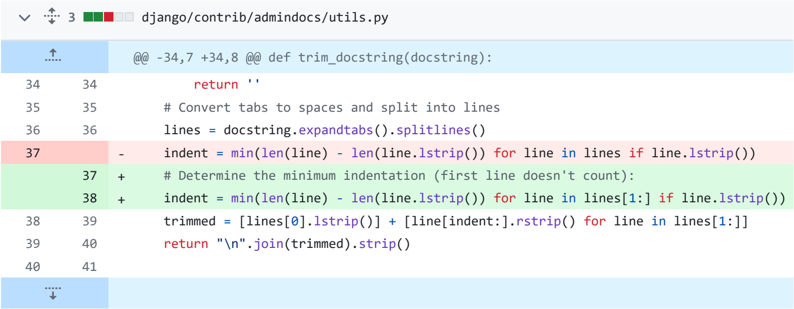
20 django/contrib/admindocs/utils.py

```
@@ -3,6 +3,7 @@
 import re
 from email.errors import HeaderParseError
 from email.parser import HeaderParser
+from inspect import cleandoc

 from django.urls import reverse
 from django.utils.regex_helper import _lazy_re_compile
@@ -24,26 +25,13 @@ def get_view_name(view_func):
     return mod_name + '.' + view_name


-def trim_docstring(docstring):
-    """
-    Uniformly trim leading/trailing whitespace from docstrings.
-
-    Based on https://www.python.org/dev/peps/pep-0257/#handling-docstring-indentation
-    """
-    if not docstring or not docstring.strip():
-        return ''
-    # Convert tabs to spaces and split into lines
-    lines = docstring.expandtabs().splitlines()
-    indent = min(len(line) - len(line.lstrip()) for line in lines if line.lstrip())
-    trimmed = [lines[0].lstrip()] + [line[indent:].rstrip() for line in lines[1:]]
-    return "\n".join(trimmed).strip()
-
-
 def parse_docstring(docstring):
     """
     Parse out the parts of a docstring.  Return (title, body, metadata).
     """
-    docstring = trim_docstring(docstring)
+    if not docstring:
+        return '', '', {}
+    docstring = cleandoc(docstring)
     parts = re.split(r'\n{2,}', docstring)
     title = parts[0]
     if len(parts) == 1:
```

3 django/contrib/admindocs/views.py

```
@@ -1,5 +1,6 @@
 import inspect
 from importlib import import_module
+from inspect import cleandoc
 from pathlib import Path

 from django.apps import apps
@@ -256,7 +257,7 @@ def get_context_data(self, **kwargs):
                     continue
                 verbose = func.__doc__
                 verbose = verbose and (
-                    utils.parse_rst(utils.trim_docstring(verbose), 'model', _('model:') + opts.model_name)
+                    utils.parse_rst(cleandoc(verbose), 'model', _('model:') + opts.model_name)
                 )
                 # Show properties and methods without arguments as fields.
                 # Otherwise, show as a 'method with arguments'.
```

Figure 15: Case from `Django` (gold) [16].



3 django/contrib/admindocs/utils.py

```
@@ -34,7 +34,8 @@ def trim_docstring(docstring):
         return ''
     # Convert tabs to spaces and split into lines
     lines = docstring.expandtabs().splitlines()
-    indent = min(len(line) - len(line.lstrip()) for line in lines if line.lstrip())
+    # Determine the minimum indentation (first line doesn't count):
+    indent = min(len(line) - len(line.lstrip()) for line in lines[1:] if line.lstrip())
     trimmed = [lines[0].lstrip()] + [line[indent:].rstrip() for line in lines[1:]]
     return "\n".join(trimmed).strip()

```

Figure 16: Case from `Django` (ours) for issue [17].

product managers, architects, project managers, etc.). This framework has achieved leading scores on the HumanEval [12] and MBPP [2], outperforming many LLMs, and researchers show its ability to complete a software establishment (e.g., a code repository to play Gomoku game), indicating that a multi-agent framework can better leverage the capabilities of LLMs in code generation tasks. Moreover, Qian et al. [43] designed ChatDev, a virtual development company simulating a human development team, which decomposes requirements into atomic tasks assigned to the developer agents. Developers mitigate the hallucination that may arise with the LLM through mutual communication and self-reflection mechanisms. Experimental results show that ChatDev can complete the establishment of some small projects (averaging no more than 5 files per project) in a relatively short time (less than 7 minutes on average). However, these works focus on the transformation from the requirements to code and overlook the code change generation during software evolution, which requires not only understanding the requirement but also dealing with the large repository.

J.3 Automatic Bug Fixing

GitHub issue resolution is a fundamental aspect of software evolution, with bug fixing being one of the most common scenarios. Fixing bugs involves both bug localization and repair. Previous researchers [65, 42] have developed methods to localize bugs before modifying the code. DreamLoc, proposed by Qi et al. [42], effectively models the characteristics of bug reports and source code files. For automatic program repair, Wong et al. [55] explored a retrieval-based method, while Ye and Monperrus [59] proposed ITER, a generation-based method for handling fault localization re-execution. Additionally, some researchers [53, 54] have combined retrieval techniques with generation models. Recently, Xia et al. [57] demonstrated that directly applying popular LLMs significantly outperforms existing APR methods, showcasing their potential for generating diverse and effective patches. Bouzenia et al. [7] introduced RepairAgent, an autonomous LLM-based agent that plans and executes bug fixes by dynamically interacting with various tools.

K Limitation

Prompt The design of prompt words may impact the performance of LLMs, thereby affecting the validity and fairness of the results [11]. While this paper focuses on innovative aspects of the proposed framework design and relies on practical guidelines for the design of prompt word templates [46] to reduce the emergence of design biases, the complete elimination of the prompt bias is extremely difficult due to the inherent biases in the dataset instances and the limitations of API resources.

Dataset The dataset contains a limited variety of software types. The evaluating dataset, SWE-bench, encompasses 12 repositories, which cover the Python programming language. However, this quantity remains insufficient compared to the diverse software projects available on GitHub. The code style, architectural design, and implementation techniques of these selected repositories, while representative, cannot fully reflect the diversity of all code repositories. In particular, the current dataset may fail to encompass some specialized fields or different programming paradigms, such as microservice architecture [66] and functional programming [29]. This limitation implies that, although our framework is designed to be independent of any specific software, the validation of its effectiveness and general applicability might be affected by this limited sample scope. Therefore, applying the findings of this paper to other code repositories may require further validation.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The main claims made in the abstract (line 5 - 16) and introduction (line 46 - 68).

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: The limitation can be found in Appendix K.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

Justification: The paper does not include theoretical results.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: The details about our framework are described in Section 3. The setup of the experimental can be found in Section 4.1.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: The data will be made public in GitHub repository: `https://github.com/co-evolve-lab/magis`.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: The experimental setting/details can be found in Section 4.1.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Statistical significance of the experiments can be found in Tab. 1, Tab. 3, etc.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [No]

   Justification: The experiments are conducted through LLMs' API rather than local compute resources.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: The paper conforms with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Positive societal impacts can be found in Section 1.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: The paper poses no such risks.

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We cite the original paper [27] that produced the dataset, SWE-bench.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: The paper does not release new assets.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.