

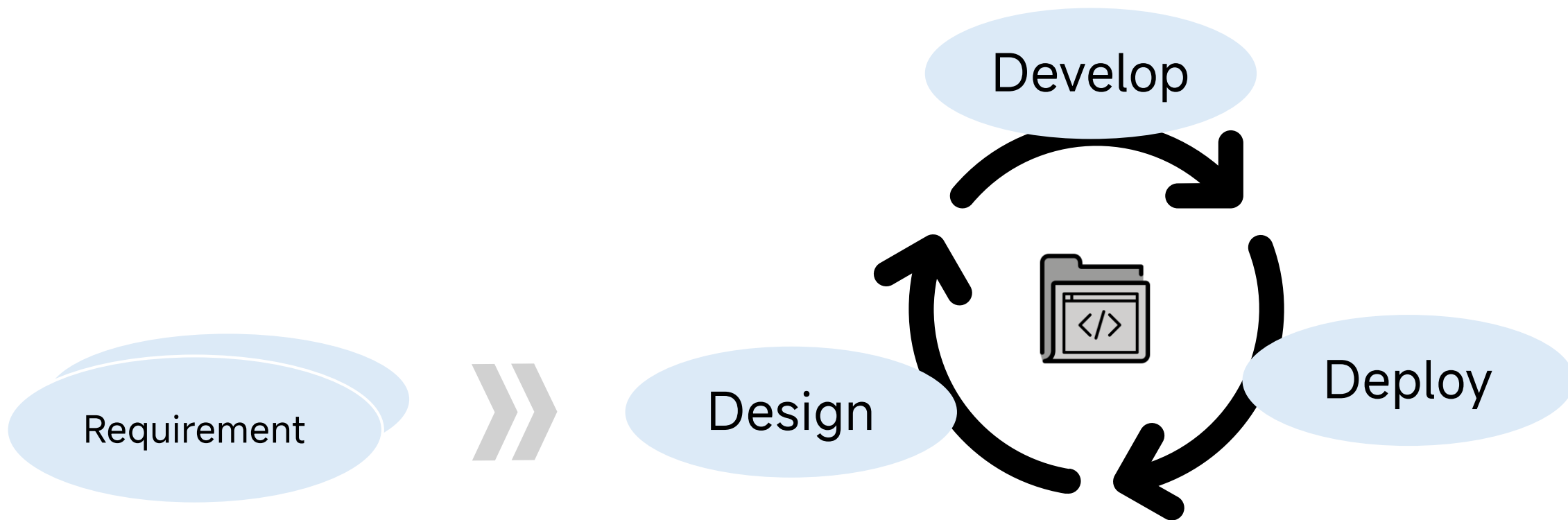
MAGIS: LLM-Based Multi-Agent Framework for GitHub Issue Resolution

Wei Tao¹, Yucheng Zhou², Yanlin Wang³, Wenqiang Zhang¹, Hongyu Zhang⁴, Yu Cheng⁵

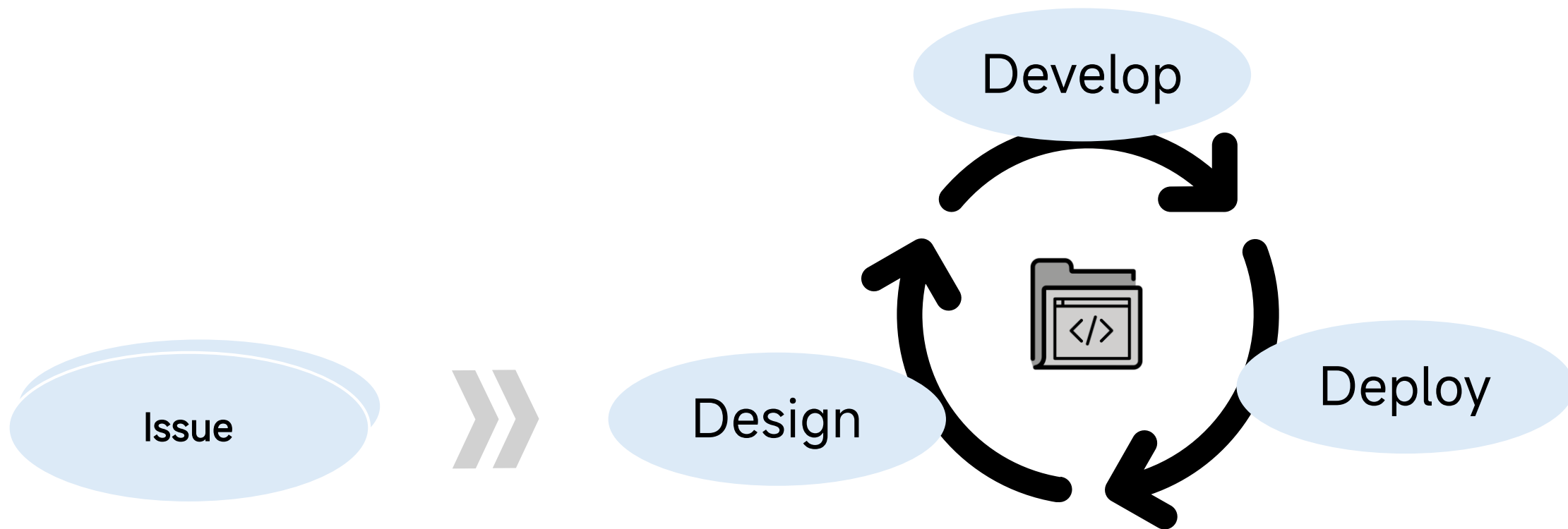
¹ Fudan University ² University of Macau ³ Sun Yat-sen University


⁴ Chongqing University ⁵ The Chinese University of Hong Kong





Software Life Cycle



Software Life Cycle at GitHub 

Model	“Oracle” Retrieval	
	% Resolved	% Apply
Claude 2	4.80	62.82
ChatGPT-3.5	0.52	21.80
GPT-4*	1.74	34.00
SWE-Llama 7b	3.01	65.52
SWE-Llama 13b	3.97	66.78



Ref: Jimenez, Carlos E., John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. "SWE-bench: Can Language Models Resolve Real-world Github Issues?." In The Twelfth International Conference on Learning Representations. 2023.

Research Questions

- *Why* the Performance of Directly Using **LLMs** to Resolve GitHub Issue is **Limited**?




Why the Performance of Directly Using LLMs to Resolve GitHub Issue is Limited?

- Locating the **Files** to be Modified
- Locating the **Lines** to be Modified
- Complexity of the **Code** Changes

Why the Performance of Directly Using LLMs to Resolve GitHub Issue is Limited?

- Locating the **Files** to be Modified
- Locating the Lines to be Modified
- Complexity of the Code Changes



Model	Max. Content 			Avg.	BM25 Recall 		
	13k	27k	50k		13k	27k	50k
Claude 2 	1.96	1.87	1.22		29.58	44.41	51.06

Why the Performance of Directly Using LLMs to Resolve GitHub Issue is Limited?

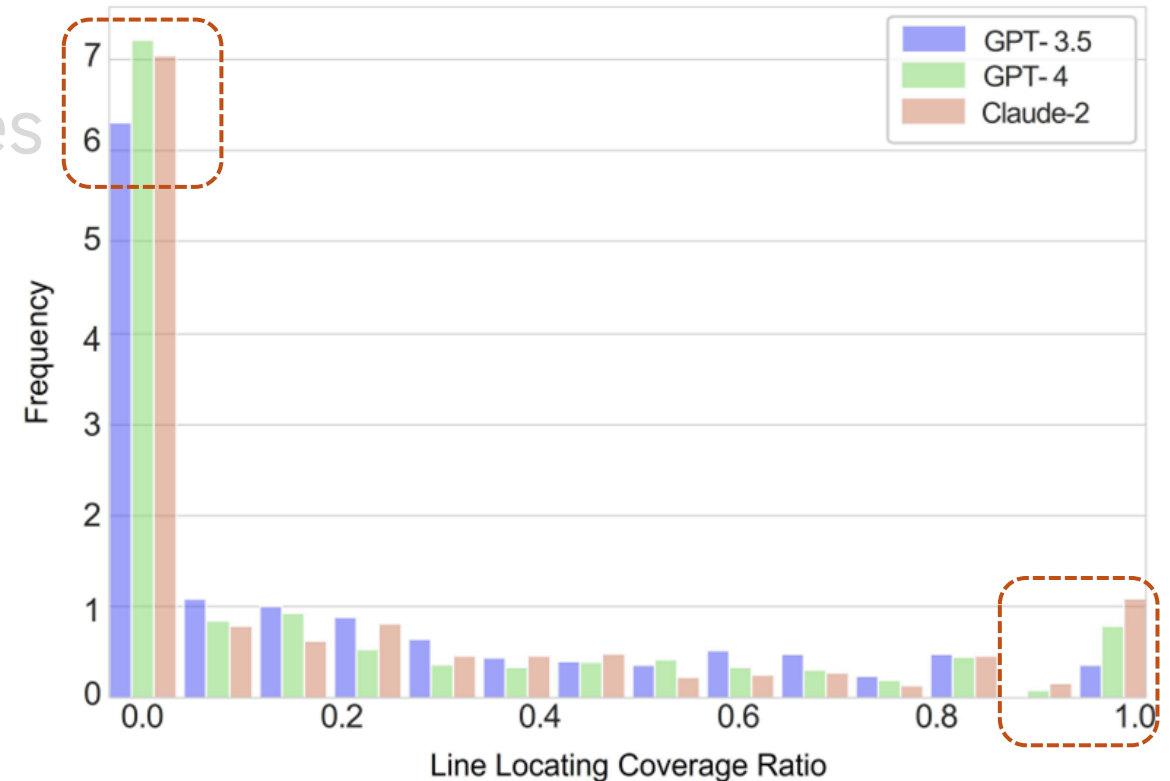
- Locating the Lines to be Modified
- Complexity of the Code Changes

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

Why the Performance of Directly Using LLMs to Resolve GitHub Issue is Limited?

- Locating the **Lines** to be Modified
- Complexity of the Code Changes

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



Why the Performance of Directly Using LLMs to Resolve GitHub Issue is Limited?

- Complexity of the Code Changes

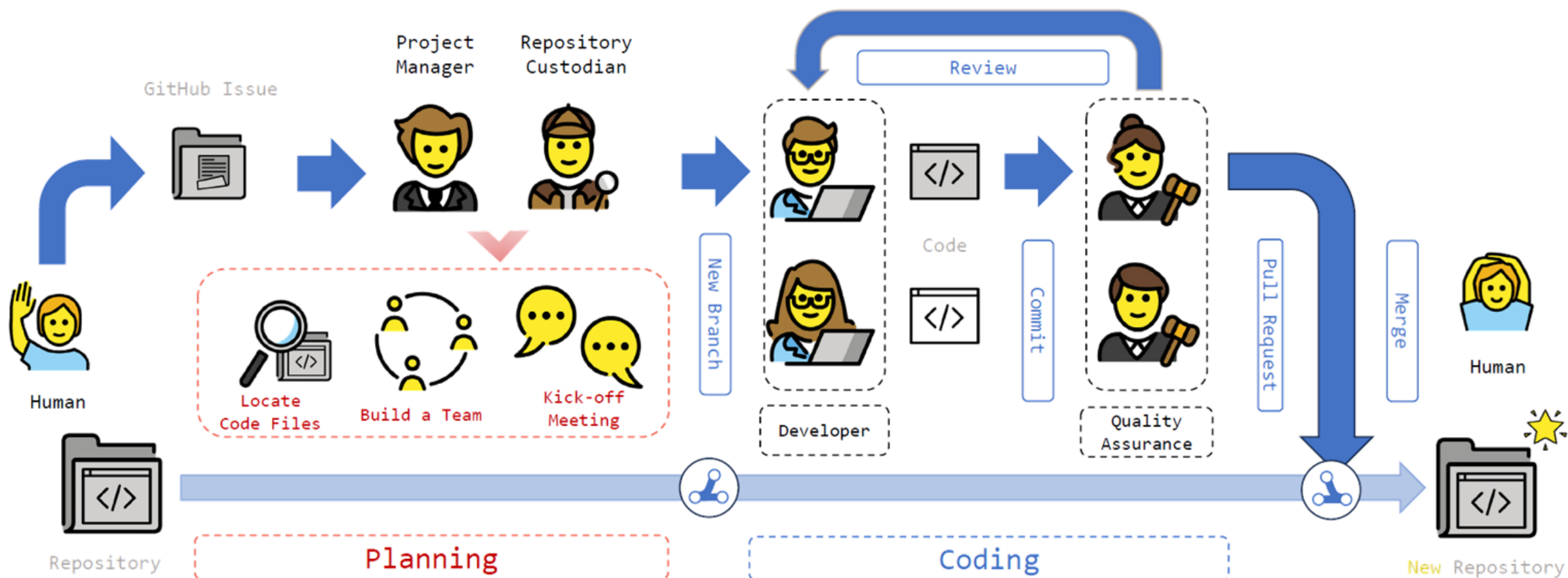
LLM	# Files	# Functions	# Hunks	# Added LoC	# Deleted LoC	# Changed LoC
GPT-3.5	-17.57*	-17.57*	-0.06*	-0.02	-0.03	-0.53*
GPT-4	-25.15*	-25.15*	-0.06	-0.10	-0.04	-0.21
Claude-2	-1.47*	-1.47*	-0.11*	-0.09*	-0.07*	-0.44*

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

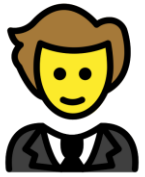
More Complex, More Difficult

How to better use LLMs for the task?

Our Framework



Agent Role Design



Project
Manager



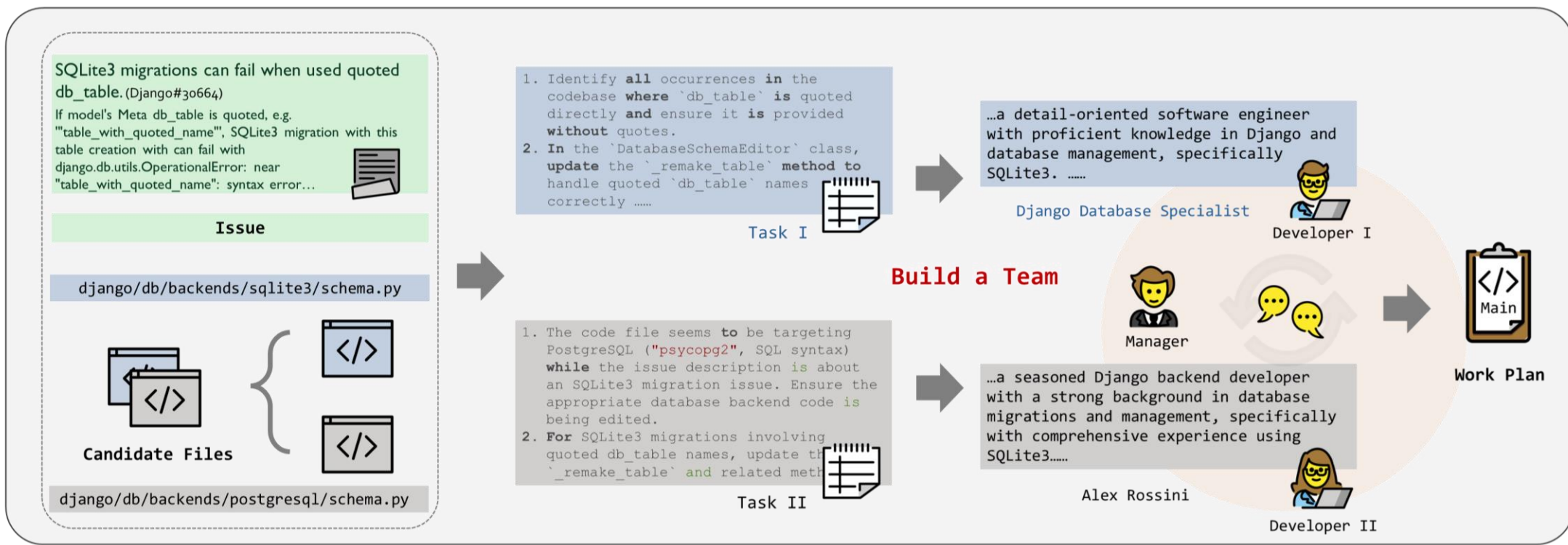
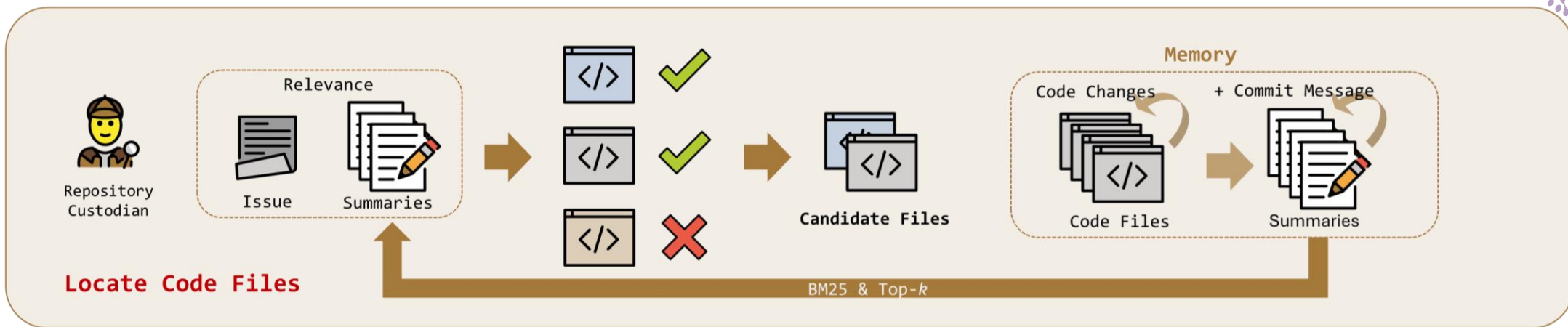
Repository
Custodian



Developer



QA Engineer



Planning



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.

Developer I

① Django Database Specialist

django/db/backends/sqlite3/schema.py

Task I

Locate Code Lines

Code Generation

TODO

Developer II

② Alex Rossini

django/db/backends/postgresql/schema.py

Task II

Code Generation

Locate Code Lines

TODO

Coding

QA Engineer I

Work Plan

1st try

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. ✓

QA Engineer II

1st try

.....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. ✗

2nd try

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. ✗

Pass All Necessary Tests ✨

Research Questions

- Why the Performance of Directly Using LLMs to Resolve GitHub Issue is Limited?
- How Effective is Our **Framework**?
- How Effective is Our Planning Process?
- How Effective is Our Coding Process?

How Effective is Our Framework?

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

How Effective is Our Framework?

```

  3  django/contrib/admindocs/utils.py
  @@ -34,7 +34,8 @@ def trim_docstring(docstring):
  34 34         return ''
  35 35         # Convert tabs to spaces and split into lines
  36 36         lines = docstring.expandtabs().splitlines()
  37 - indent = min(len(line) - len(line.lstrip()) for line in lines if line.lstrip())
  37 + # Determine the minimum indentation (first line doesn't count):
  38 + indent = min(len(line) - len(line.lstrip()) for line in lines[1:] if line.lstrip())
  38 39         trimmed = [lines[0].lstrip()] + [line[indent:].rstrip() for line in lines[1:]]
  39 40         return "\n".join(trimmed).strip()
  40 41
  
```

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

```

  20  django/contrib/admindocs/utils.py
  @@ -3,6 +3,7 @@
  3 3  import re
  4 4  from email.errors import HeaderParseError
  5 5  from email.parser import HeaderParser
  6 + from inspect import cleandoc
  6 7
  7 8  from django.urls import reverse
  8 9  from django.utils.regex_helper import _lazy_re_compile
  @@ -24,26 +25,13 @@ def get_view_name(view_func):
  24 25         return mod_name + '.' + view_name
  25 26
  26 27
  27 - def trim_docstring(docstring):
  28 -     """
  29 -     Uniformly trim leading/trailing whitespace from docstrings.
  30 -
  31 -     Based on https://www.python.org/dev/peps/pep-0257/#handling-docstring-indentation
  32 -     """
  33 -     if not docstring or not docstring.strip():
  34 -         return ''
  35 -     # Convert tabs to spaces and split into lines
  36 -     lines = docstring.expandtabs().splitlines()
  37 -     indent = min(len(line) - len(line.lstrip()) for line in lines if line.lstrip())
  38 -     trimmed = [lines[0].lstrip()] + [line[indent:].rstrip() for line in lines[1:]]
  39 -     return "\n".join(trimmed).strip()
  40 -
  41 -
  42 28  def parse_docstring(docstring):
  43 29     """
  44 30     Parse out the parts of a docstring. Return (title, body, metadata).
  45 31     """
  46 - docstring = trim_docstring(docstring)
  46 + if not docstring:
  47 +     return '', '', {}
  47 + docstring = cleandoc(docstring)
  47 35     parts = re.split(r'\n(2,}', docstring)
  48 36     title = parts[0]
  49 37     if len(parts) == 1:
  
```

```

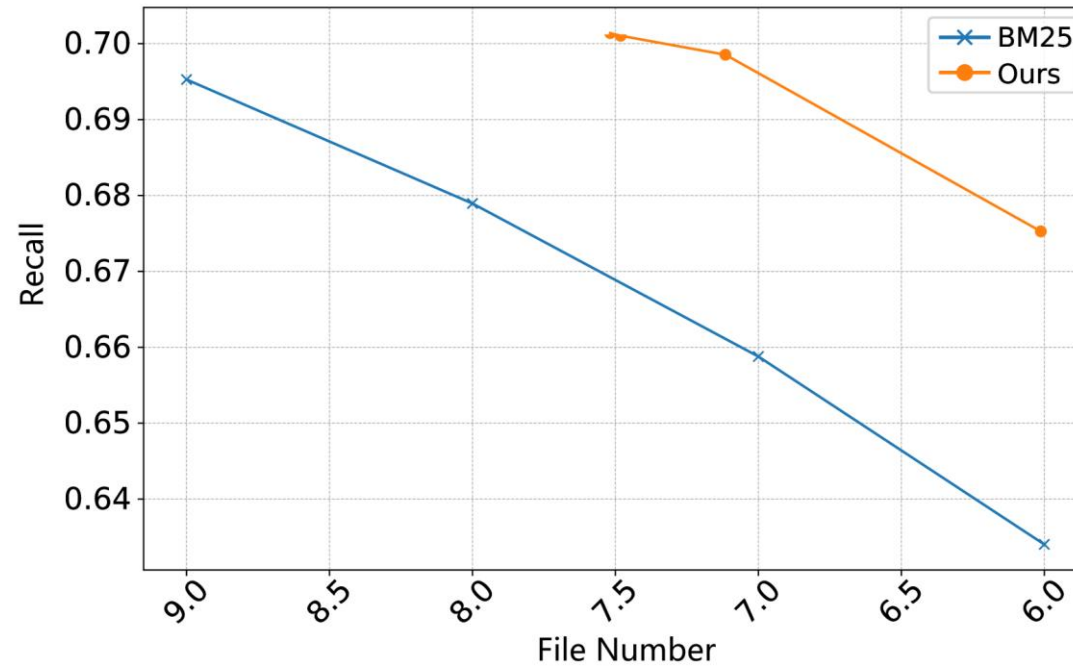
  3  django/contrib/admindocs/views.py
  @@ -1,5 +1,6 @@
  1 1  import inspect
  2 2  from importlib import import_module
  3 + from inspect import cleandoc
  3 4  from pathlib import Path
  4 5
  5 6  from django.apps import apps
  @@ -256,7 +257,7 @@ def get_context_data(self, **kwargs):
  256 257         continue
  257 258         verbose = func.__doc__
  258 259         verbose = verbose and (
  259 -         utils.parse_rst(utils.trim_docstring(verbose), 'model', _('model:') + opts.model_name)
  260 +         utils.parse_rst(cleandoc(verbose), 'model', _('model:') + opts.model_name)
  260 261     )
  261 262     # Show properties and methods without arguments as fields.
  262 263     # Otherwise, show as a 'method with arguments'.
  
```

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

Research Questions

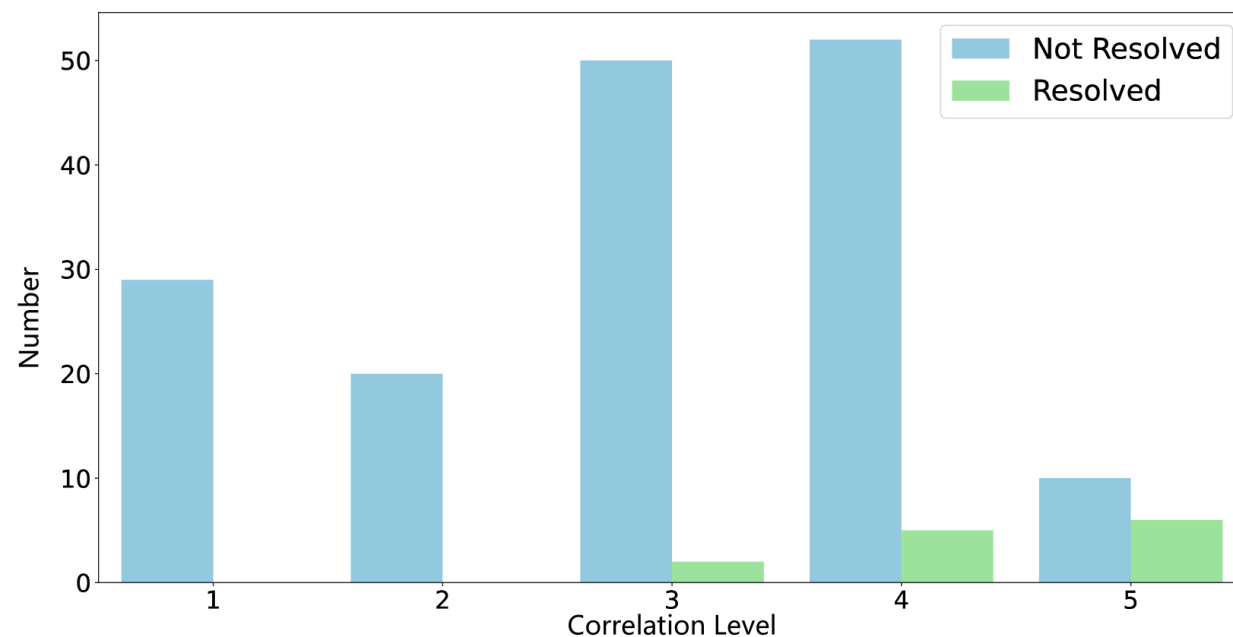
- Why the Performance of Directly Using LLMs to Resolve GitHub Issue is Limited?
- How Effective is Our Framework?
- How Effective is Our **Planning** Process?
- How Effective is Our Coding Process?

How Effective is Planning Process?



Comparison of recall scores between Ours and BM25.

How Effective is Planning Process?

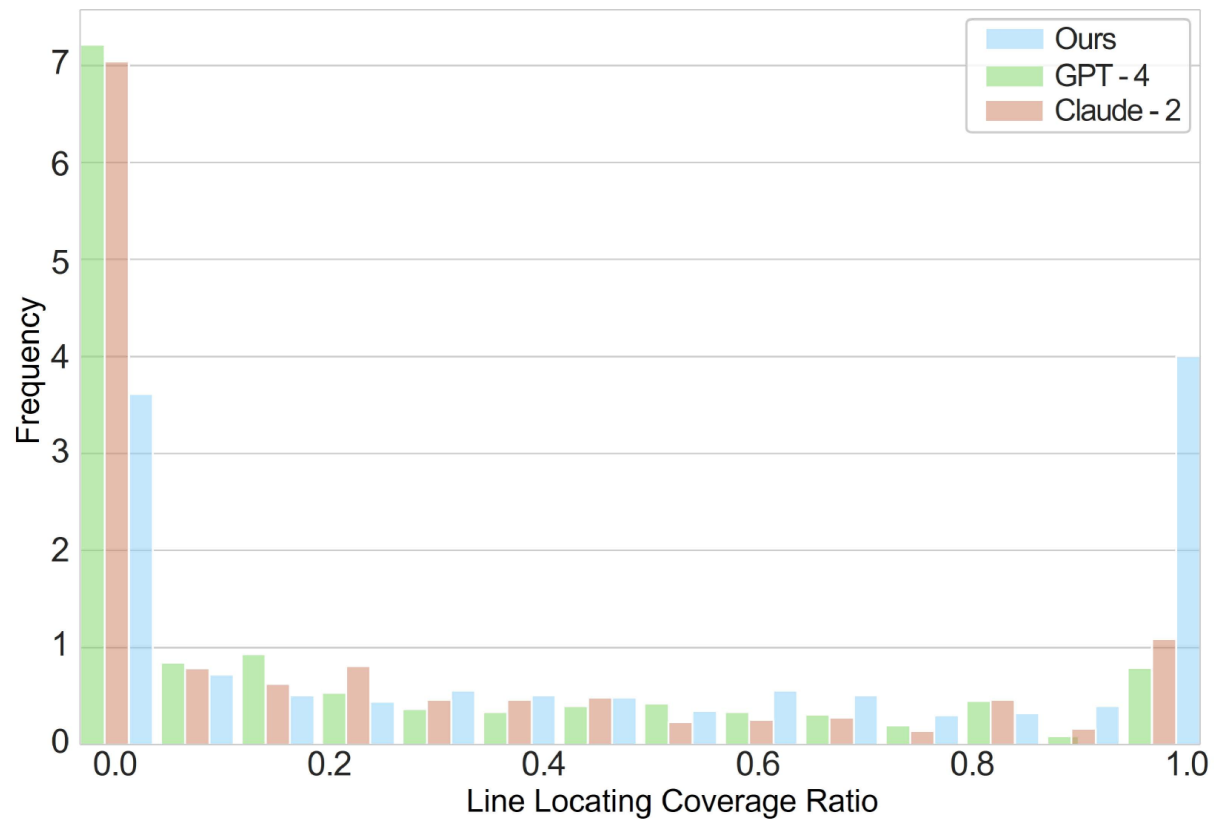


Distribution of the correlation score between the generated task description and the reference code change.

Research Questions

- Why the Performance of Directly Using LLMs to Resolve GitHub Issue is Limited?
- How Effective is Our Framework?
- How Effective is Our Planning Process?
- How Effective is Our **Coding** Process?

How Effective is Coding Process?



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

How Effective is Coding Process?

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).

Conclusion

- Empirical analysis of LLMs in resolving GitHub issues.
- A novel LLM-based multi-agent framework.
- Experiments & Analysis.

THANKS!