Issue Localization via LLM-Driven Iterative Code Graph Searching

Supplementary Material

A. Call Graph in Code Repository

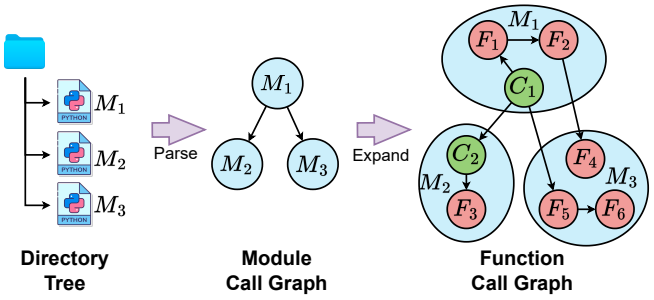

Fig. 5. An example of call graph construction.

As shown in Figure 5, in the root directory, module M_1 imports modules M_2 and M_3, and their invocation relationships are represented in the module call graph. Furthermore, M_1 contains class C_1 and functions F_1 and F_2, allowing the nodes in the module call graph to be expanded, as illustrated in the function call graph on the right.

B. Function Call Graph Searching Algorithm

We provided the Algorithm 1 described in Section III-A2.

Algorithm 1 Function Call Graph Searching with Pruning

Input: $\mathcal{G}_F = (\mathcal{V}_f, \mathcal{E}_f)$ ▷ Function call graph
Input: maxIter ▷ Maximum of iterations
Output: contextNodes ▷ The relevant functions or classes

```
contextNodes={};
visableNodes=initStartPoints(V_f);  ▷ Pre-select by LLM
counter=0;
while counter<maxIter do  ▷ Start Searching
    targetNode=Seacher(visableNodes);  ▷ Search Step
    if targetNode == null then
        break
    end if
    nodeContext=getCode(targetNode);
    pruningFlag = Pruner(nodeContext);
    if pruningFlag then  ▷ Pruning context
        visableNodes = visableNodes - targetNode ;
    else
        nextNeighbors = getNeighbors(targetNode) ;
        visableNodes = visableNodes ∪ nextNeighbors ;
        contextNodes = contextNodes ∪ targetNode ;
    end if
    counter=counter+1;
end while
return contextNodes;
```

C. Examples Demonstrating the Effectiveness of Key Components in IGSIL

To better understand the performance differences among the four variants in the ablation study, we manually inspect the experimental results and select two highly representative examples.

We find that without reflective alignment, IGSIL not only triggers more "*KeyErrors*", making it impossible to parse the localization results, but also loses the potential for result re-ordering and error correction. As shown in Figure 6, for the case of Django-12908, all output paths produced by IGSIL without reflective alignment are incomplete, containing only the final file name without the full path starting from the root directory "`django/`", making it impossible for the system to parse them. As shown in Figure 7, for Django-13315, IGSIL without reflective alignment fails to reorder and correct the retrieved files, causing the target file "`django/forms/models.py`" to be excluded from the top five candidate files, thus directly losing the correct search space.

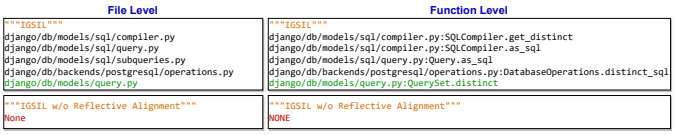

Fig. 6. An example of localization failure after removing reflective alignment from Django-12908.

Without the module call graph, IGSIL exhibits an overly narrow search space at the file level. As shown in Figure 7, the LLM has no way of obtaining information about the target file "`django/forms/models.py`" from any source, thereby restricting the search space to the issue description and ultimately causing subsequent localization failure.

Without iterative search, although the correct search space is identified, the function signature information within the file skeleton is insufficient to support the LLM in making accurate judgments. This approach loses the depth of search, forcing the LLM to match function signatures based solely on the issue description, leading to the selection of "`django/db/models/fields/related.py:ForeignKey.formfield`".

Without pruning, we observed that the LLM tends to conduct searches in multiple directions. It sequentially queries almost all class nodes in "`django/db/models/fields/related.py`" and "`django/forms/models.py`". This not only consumes a large portion of the context window but also exhausts the number of allowed iterations, causing the algorithm to terminate before it reaches the exploration of "*apply_limit_choices_to_to_formfield*".

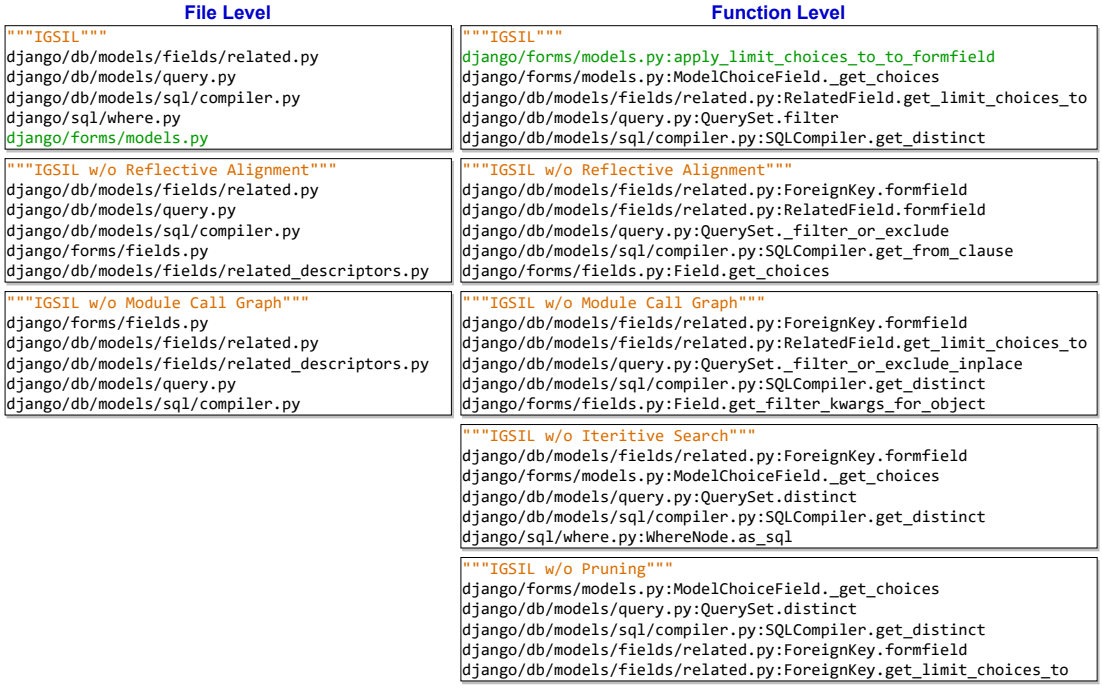

Fig. 7. An example where the removal of any core component leads to localization failure from Django-13315.

TABLE X
LOCALIZATION RESULTS UNDER DIFFERENT FAMILIES OF LLMS ON SWE-BENCH LITE MINI AND VERIFIED MINI.

Method	File-level					Function-level					
	Top-1	Top-3	Top-5	MAP	MRR	Top-1	Top-3	Top-5	MAP	MRR	ER
SWE-bench Lite mini											
Deepseek-v3-0324											
Agentless-FL	**0.78**	**0.84**	0.86	0.811	0.811	0.28	0.42	0.42	0.331	0.339	**0.00**
OrcaLoca	0.76	0.76	0.76	0.76	0.76	0.44	0.64	0.64	0.527	0.540	0.08
LocAgent	0.72	0.78	0.78	0.747	0.747	0.14	0.40	0.54	0.276	0.292	0.16
IGSIL	**0.78**	**0.84**	**0.90**	**0.813**	**0.813**	**0.64**	**0.67**	**0.78**	**0.659**	**0.697**	**0.00**
GPT-4o-2024-0806											
Agentless-FL	**0.76**	0.86	**0.92**	**0.816**	**0.816**	0.30	0.66	0.70	0.453	0.471	**0.00**
OrcaLoca	0.66	0.70	0.70	0.680	0.680	0.28	0.50	0.58	0.387	0.392	0.02
LocAgent	0.62	0.76	0.76	0.683	0.683	0.10	0.32	0.44	0.218	0.224	0.16
IGSIL	0.70	**0.88**	**0.92**	0.785	0.785	**0.56**	**0.70**	**0.74**	**0.604**	**0.630**	**0.00**
SWE-bench Verified mini											
Deepseek-v3-0324											
Agentless-FL	**0.80**	0.86	**0.92**	**0.818**	0.836	0.46	0.64	0.64	0.494	0.549	**0.00**
OrcaLoca	0.74	0.80	0.80	0.740	0.770	0.40	0.76	0.76	0.507	0.507	0.04
LocAgent	0.68	0.78	0.80	0.706	0.732	0.30	0.54	0.60	0.390	0.524	0.14
IGSIL	0.78	**0.92**	**0.92**	0.815	**0.847**	**0.66**	**0.76**	**0.76**	**0.636**	**0.703**	**0.00**
GPT-4o-2024-0806											
Agentless-FL	**0.76**	0.84	0.86	0.782	0.802	0.58	**0.76**	**0.78**	0.599	0.669	**0.00**
OrcaLoca	0.70	0.74	0.76	0.700	0.725	0.36	0.60	0.62	0.459	0.472	0.04
LocAgent	0.50	0.56	0.56	0.517	0.527	0.24	0.38	0.46	0.290	0.324	0.40
IGSIL	0.74	**0.88**	**0.90**	**0.791**	**0.807**	**0.60**	**0.76**	**0.78**	**0.620**	**0.672**	**0.00**

D. RQ4's Results on Different LLMs.

Table X shows the complete results of experiments run on different families of LLMs. IGSIL outperforms all baseline methods at the function level, indicating that IGSIL generalizes well to different families of LLMs. In addition, we observed that Agentless-FL demonstrates the strongest Top-1 file-level localization, which may be because IGSIL expands the search space at the file level, resulting in some additional modules being included in the localization results. Agentless-FL's results on SWE-bench Verified mini are nearly on par with IGSIL, especially when using GPT-4o. This may be because the issue descriptions in SWE-bench Verified are of higher quality and contain information about the target function of the task.

E. Prompt Templates

Please look through the following GitHub problem description and Repository structure and provide a list of files that one would need to edit to fix the problem.
I have already find 5 relevent files. Accrording to the import relations, construct the call graph first.
Then, Rank them again and reflect the result.

GitHub Problem Description
{problem_statement}

###

Repository Structure
{structure}

###

Files To Be Ranked
{pre_files}

###

Import Relations
{import_content}
###

Please only provide the full path and return top 5 files.
The returned files should be separated by new lines ordered by most to least important and wrapped with ```
For example:
```
file1.py
file2.py
file3.py
file4.py
file5.py
```
Note: file1.py indicates the top-1 file, file2.py indicates the top-2 file, and so on. Do not include test files.

Fig. 9. Prompt for Module Call Graph Enhanced Search Space Reduction.

You will be presented with a bug report and tools (functions) to access the source code of the system under test (SUT).
Since the modification is based on the code repository, the modified locations may include files, classes, and functions, and the modifications may be in the form of addition, deletion, or update.
Your task is to locate the top-5 most likely culprit locations based on the bug report and the information you retrieve using given functions.
Function calls you can use are as follows:
* get_code_of_class('file_name', 'class_name') -> Get the code of a specified class in the given file and python project. 'file_name' -> The name of the file. 'class_name' -> The name of the class. *
* get_code_of_class_function('file_name', 'class_name', 'func_name') -> Get the code of a specified function in the given class, file, and python project. 'file_name' -> The name of the file. 'class_name' -> The name of the class. 'func_name' -> The name of the function. *
* get_code_of_file_function('file_name', 'func_name') -> Get the code of a specified function in the given file and python project. 'file_name' -> The name of the file. 'func_name' -> The name of the function. *
* exit() -> Exit function calling to give your final answer when you are confident of the answer. *
You have {max_try} chances to call function.
The bug report is as follows:
```
### GitHub Problem Description ###
{problem_statement}

###
Let's locate the faulty file step by step using reasoning and function calls.
I have pre-identified top-5 files that may contain bugs. There stuctures are as follows:
{bug_file_list}
The formal parameter 'file_name' takes the value in "file:"
The formal parameter 'ckass_name' takes the value in "class:"
The formal parameter 'func_name' takes the value in "static functions:" and "class functions: "
Avoid making multiple identical calls to save overhead.
You must strictly follow the structure I give to call different tools.
For static functions, you can use 'get_code_of_file_function', and for class functions, you can use 'get_code_of_class_function'.
In order to locate accurately, you can pre-select {pre_select_num} locations, then check them through function calls, and finally confirm {top_n} file names.
Don't make the first function call in this message.

Fig. 10. Prompt for Iterative Search.

You will be presented with a bug report with repository structure to access the source code of the system under test (SUT).
Your task is to locate the most likely culprit functions/classes based on the bug report.
<bug report>
{problem_statement}
</bug report>

Here is a result of a function/class code retrived by '{content}'.
Please check if the code is related to the bug and if the code should be added into context.
<code>
{function_retval}
</code>
Return True if the code is related to the bug and should be added into context, otherwise return False.
Since your answer will be processed automatically, please give your answer in the format as follows.
The returned content should be wrapped with ```.
```
True
```
or
```
False
```

Fig. 11. Prompt for Pruning.

Here is a localization result, but it seems not in the correct format. Please correct it.
The returned files should be separated by new lines ordered by most to least important and wrapped with ```
This is an example of expected output:
```
sklearn/linear_model/__init__.py
sklearn/base.py
```

Please help me corrct the following result.
{res}

Fig. 12. Prompt for Reflective Alignment
```

You will be presented with a bug report with repository structure to access the source code of the system under test (SUT).
Your task is to locate the top-5 most likely culprit files based on the bug report.
The bug report is as follows:
```
### GitHub Problem Description ###
{problem_statement}

###

### Candidate Files ###
{structure}

###
```
Let's locate the faulty file step by step using reasoning.
In order to locate accurately, you can pre-select {pre_select_num} files, then check them through function calls, and finally confirm {top_n} file names.
Based on the available information, reconfirm and provide complete name of the top-5 most likely culprit files for the bug.
Since your answer will be processed automatically, please give your answer in the format as follows.
The returned files should be separated by new lines ordered by most to least important and wrapped with ```.
```
file1.py
file2.py
file3.py
file4.py
file5.py
```
Replace the 'file1.py' with the actual file path.
For example,
```
sklearn/linear_model/__init__.py
sklearn/base.py
```

Fig. 8. Prompt for Pre-Selection Code Files.