Can Timing Localize Agent Failures?

Incorperating a temporal dimension into spectrum-based fault localization for LLM-based multi-agent systems

Author: Hein Schouwenaars (hmschouwenaars@tudelft.nl)
Supervisors: Burcu Kulahcioglu Ozkan, Annibale Panichella, Zahra Seyedghorban

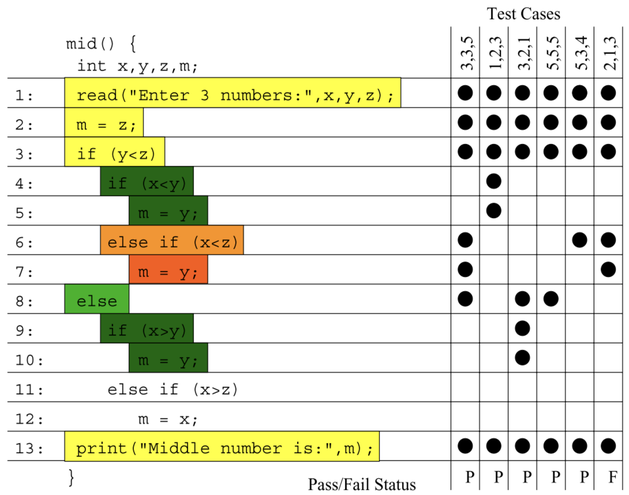
1. Introduction

Localizing faults in Large Language Model-based Multi Agent System can not yet be done reliably because of their indeterminism.

Spectrum based fault localization (SBFL) is a fault localization technique that assigns a suspiciousness ranking to program element based on the number of executions passing and failing tests.

Applying this to LLM-MAS requires redefining program elements; we will test grouping agent actions into windows based on temporal order.

1.1 *SBFL in classical software*

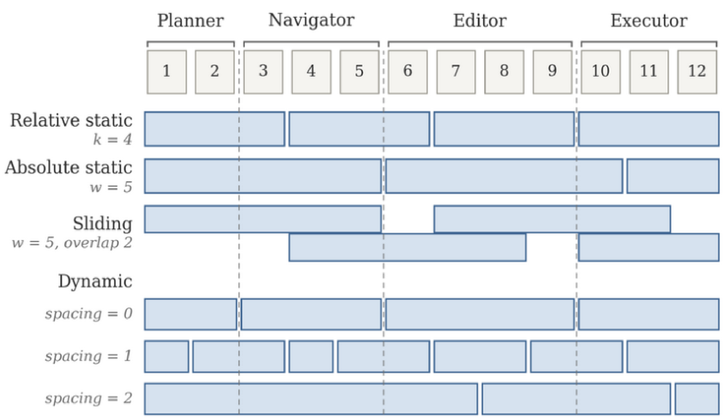


2. Research Question

Four windowing techniques are proposed:
Absolute static, relative static, sliding, and dynamic windows. Seen in figure 2.1.

Research Question: To what extend do temporal static window, sliding window, and dynamic window based spectra differ in their spectrum-based fault localization accuracy when applied to LLM-based multi-agent systems?

2.1 *Four windowing techniques*



3. Method

1. Run HyperAgent on three SWE-Bench tasks and collect execution logs. Seen in figure 3.1
2. Parse tool usage and classify each step's messages into 6 types, forming agent–action pairs.
3. Split each run into windows per strategy; group steps into agent–action–window triples.
4. Count triple frequencies across passing and failing logs. Seen in figure 3.2
5. Apply 5 suspiciousness formulae to rank the triples.
6. Collapse the window axis; rank distinct agent–action pairs.
7. Label ground-truth faults with an LLM-as-judge.
8. Compute top-k accuracy for each windowing strategy.
9. Tune each strategy's parameters and compare accuracies.

3.1 *Execution logs*

Task	# Logs	Pass %
pallets__flask-5014	165	33.3
psf__requests-1142	81	70.4
psf__requests-1766	96	49.0

3.2 *Example window participation relative static windows (k = 3)*

	Failed (n=49)			Passed (n=47)		
Agent / action	w_1	w_2	w_3	w_1	w_2	w_3
Executor Interpreter / report	0	58	199	6	122	382
Editor Interpreter / report	61	167	58	94	242	40
Navigator Interpreter / report	213	72	53	233	56	17
Inner-Navigator-Assistant / exploration	195	69	39	210	49	12

4. Results

The results (figure 4.1) show relative static windowing performs best. Reaching 52.8% found at k = 10. It is the only windowing technique that outscores the baseline over every k.

The other techniques do not outscore the baseline consistently, and. Sliding windows never peforms worse than the baseline, but it's gains are marginal.

No stategy comes close to the ceiling over all k values. Relative static windowing comes closest at k = 1.

4.1 *Top-k localization score*

Score (%)	k=1	k=3	k=5	k=7	k=10
Windowing strategy (Kulczynski2)					
Static (Absolute, w=3)	0.0	25.0	25.0	33.3	49.9
Static (relative, k=39)	**27.8**	**35.2**	**36.9**	**39.6**	**52.8**
Sliding (w=1, O=38)	10.0	25.0	29.2	32.6	35.3
Dynamic (*spacing*=2)	10.0	10.0	25.0	29.2	44.7
Random (weighted)	2.9	8.7	14.5	20.2	28.7
Baseline (one window)	1.4	5.6	29.2	31.9	34.5
Ceiling (best possible)	31.9	63.4	75.1	82.8	91.4

Figure 4.1: Top-k fault-localization score (%), averaged across the three tasks. *Random*: Random suspicousness ranking, weighted by occurance count. *Baseline*: relative windowing with k = 1, using the best formula averaged over k (Kulczynski2). *Ceiling*: best attain- able. Best per column in **bold**.

5. Conclusion

Only relative static windowing reliably outpeforms the baseline, suggesting it can be used to aid fault localization.
The other strategies do not suggest to consistently improve fault localization by a reasonable margin
The action—agent—window spectrum reached 52.8% accuracy at it's peak, suggesting the spectrum cannot reliably localize faults in LLM-MAS.

6. Future work

Future work should include research into new spectra, possibly containing relative static windowing.
Another promising fault localization technique is LLM-as-a-judge, which showed great potential for LLM-MAS fault localization. Thus LLM-as-a-judge must also be explored further.
Finally, there is a need for a trace dataset containing passing and failing runs over multiple LLM-MAS frameworks and tasks with human annotated failures