CS846 Week 3 Summary Review

Amaan Ahmed

Paper: Explaining GitHub Actions Failures with Large Language Models: Challenges, Insights, and Limitations [1]

Problem Being Solved

GitHub Actions is widely used as a CI/CD tool to automate testing, building, and deployment. While powerful, its workflows fail quite often, not only due to bugs in the code but maybe because of missing dependencies, misconfigured environment variables, or errors in the configuration files themselves. When failures occur, developers are left with massive log files that are meant to explain what went wrong and these logs are highly technical, and rarely user-friendly. Consequently, developers often need to sift through hundreds of lines just to find the cause of the failure. The result is therefore, wasted time, effort and frustration before debugging can even begin to start.

New Idea

The paper proposes the use of large language models to parse long unstructured logs and explain failures in clear, actionable natural language summaries. If successful, this could drastically reduce developer effort and improve productivity.

To explore this idea, the authors start with defining what constitutes a good or useful explanation. They choose four attributes and define them as follows:

- **Correctness:** Are the explanations technically accurate?
- **Clarity:** Are the explanations easy to understand and follow?
- **Conciseness:** Do the explanations avoid unnecessary details and stick to essentials?
- **Actionability:** Do the explanations provide clear next steps to resolve the issue?

These attributes map directly to the three research questions (RQs):

- **RQ1: To what extent do LLMs correctly describe the context of GA run failures according to developers?**
  This question addresses **correctness**; are the explanations technically sound and free from misleading information?
- **RQ2: To what extent do developers find LLM generated explanations for GA run failures clear and concise?**
  This question addresses **clarity** and **conciseness**; how accessible are the explanations? Do the explanations contain only essential information?
- **RQ3: To what extent are the descriptions of GA run failures considered actionable by developers?**
  This question addresses **actionability**; do the explanations provide specific and relevant information which developers can implement into a solution easily?

To then address these questions, the authors designed a survey-based study combining close-ended Likert-scale questions for **correctness, clarity**, and **conciseness** with two open-ended questions to collect free-text feedback and assess **actionability**. This mix provided a balance of both structured, quantitative data and qualitative insights, minimizing the authors' own bias and involvement in data collection.

Next, to deliver this survey in a consistent and unbiased way, the authors built their own platform, **LogExp**. This meant that every participant viewed logs, explanations and answered the survey questions in the exact same manner, ensuring consistency that reduced presentation bias and streamlined the process.

For the choice of model to generate the explanations, the authors ran a pilot study testing LLaMA2, LLaMA3, and Mixtral under different prompting strategies (zero-shot, one-shot, few-shot). They settled on **LLaMA3** with **one-shot prompting**, finding it produced the best balance of accuracy and simplicity.

Results-wise, for ***RQ1*** **(correctness)**, results were strongly positive. Over 80% of participants agreed that explanations were accurate, logically coherent, and precise enough to diagnose failures. For ***RQ2*** **(clarity and conciseness)**, again $\sim$ 80% rated explanations as clear and easy to follow, while conciseness was slightly weaker ($\sim$ 74.5%). For ***RQ3*** **(actionability)**, results were mixed and more nuanced. The authors derived five sub-attributes of **actionability** from qualitative responses:

- **Clarity of Explanation:**
  If the explanation was confusing or used vague language, it couldn't really help them act on it.
- **Actionable Guidance:**
  A good explanation might point to installing a missing dependency or updating a configuration, instead of simply repeating the error message.
- **Specificity of Content:**
  Generic statements were useless, while context-specific ones were appreciated.
- **Contextual Relevance:**
  Including details like defined dependencies or links to external documentation made explanations more helpful.
- **Conciseness:**
  Explanations should be brief and to the point; avoid fluff and unnecessary details.

Positive Points

- I liked the paper's potential impact. LLMs can greatly reduce human effort when dealing with error logs, and this research could inspire similar efforts on other kinds of large, unstructured data.
- I thought the paper took a very systematic approach. The authors first defined attributes to judge the explanations and these attributes then tied in seamlessly with the research questions. It made the whole study structured and easy to follow.
- I also liked how the evaluation was consistent and balanced. By mixing close-ended and open-ended questions, the authors reduced their own bias while still collecting richer insights. A big strength for me was the LogExp tool. Building their own platform showed commitment to fairness and consistency.
- The inclusion of a dedicated section for defending the validity of the paper was interesting to see. The authors actually

thought carefully about the limitations and possible criticisms of their design. For example, they acknowledged issues like the small response rate and other ways the survey could have been designed. But at the same time, they gave clear reasoning for why they made the choices they did. To me, this section showed self-awareness and honesty, which makes the paper much stronger.
- Finally, I liked that the authors provided a replication package with raw data and setup files. This makes it easier to replicate and verify the work done, increasing transparency and reliability.

Negative Points

- The response rate was extremely low. Out of 811 developers contacted, only 31 responded, and only those surveys that were at least 70% complete were included. This makes the final sample size not only small but also somewhat indeterminate, since we don't know exactly how many full responses were analyzed. Greater clarity here would have strengthened the results.
- The paper frames the problem as massive, unstructured logs that are hard to parse. Yet the examples shown in the paper were relatively simplified, not the overwhelming 'log swamps' developers typically face. It is also unclear whether the LLMs were run on entire log files or just the curated portions displayed in LogExp. More detail on this process would have been valuable.
- The participant pool may have biased the results. The study specifically recruited skilled developers with prior experience handling GitHub Actions failures, and some had contributed to projects from which the failures were sourced. This raises the possibility that participants already knew the root causes before reading the explanations, inflating correctness and actionability scores. Including less experienced developers, or experienced developers who had not worked on the failures, could have provided more balanced insight into whether the explanations truly aid understanding.

Future Work

- A more representative pool of developers could give stronger evidence and including less experienced developers could provide the opportunity to learn from their unique feedback too.
- Another direction would be to test fine-tuned LLMs that are trained specifically on CI/CD logs. The models used here were general-purpose, which is why they sometimes performed poorly. A specialized model might improve correctness and actionability.
- It would be interesting to extend this research beyond GitHub Actions. Other CI/CD tools like Jenkins or Azure Pipelines also produce massive logs. Testing across different platforms would show whether this approach generalizes.

Rating

4, great and intuitive application of LLMs to make life easier.

Discussion Points

- **Can LLMs be trusted for this task if they sometimes give confident but wrong or misleading explanations?**
How do we mediate this? What sort of manual intervention can help but maintain reduced effort?
Brian mentioned the inclusion of a feature that could show the 'chain-of-thoughts' behind a LLM's explanation. This would increase transparency as well as allow the developer to evaluate the LLM's thought process and see if, and where, the LLM may have made a mistake. Jacie proposed the use of SWE-agent to assist developers with this issue, while Tongwei suggested leveraging multi-models for cross-verification.
- **Is conciseness more important than actionability?**
What trade-offs can we feasibly undertake here?
I was of the opinion that these two attributes were a bit conflicting since if we have a situation where an LLM-generated explanation contains a lot of actionable steps to solve the problem, this reduces conciseness. Vice versa, if we reduce the number of steps to improve conciseness, actionability may diminish. The ideal solution would be to find a balance between the two but how exactly? To this, Youssef suggested the provision of a concise suggestion of steps to the developers but also providing the developer with an option to expand for more detail. Having this option would cater to both novice and experienced developers. Felix agreed that different developers with different levels of experience would naturally have different needs and preferences. He remarked that the trade-off between conciseness and actionability depends on the complexity of the job or steps suggested to resolve the failure. Simpler jobs requiring short scripts would demand conciseness while more complex jobs would require elaboration for actionability. Asim commented that solving GA run failures is not for everyone and only experienced developers would be needing the use of such a tool. Thus, his opinion was that this trade-off really depended on a defined goal for the tool; if experienced developers are to use this tool, then surely their needs are more important and such developers often value conciseness.
- **How do we balance the risk of developer over-reliance on LLMs versus their productivity benefits?**
Developers already employ LLMs to generate code, will this affect their debugging skills as well? What skills would developers require then?
Similar to last week's discussion on the evolving role of a software developer, this point questioned how skills will shift if even debugging is delegated to LLMs. In my view, one thing that makes a developer strong is their debugging skills and ability to look through lines of code (or text) and figure out what's wrong. Developers already utilise LLMs in generating code so if LLMs take over debugging as well, what will remain for us, and what new skills must we develop? Jacie noted that the professional world demands skills and adaptability, where developers are trained to have skills sufficient to perform well in this world. The introduction of such tools would then mean developers would need to learn how to decide when, and how, to use which tools. Kevin agreed, adding that as technology evolves and times change, so must the skills of developers, with prompt engineering increasingly becoming an essential competency.

References

[1] Pablo Valenzuela-Toledo, Chuyue Wu, Sandro Hernandez, Alexander Boll, Roman Machacek, Sebastiano Panichella, and Timo Kehrer. 2025. Explaining GitHub Actions Failures with Large Language Models: Challenges, Insights, and Limitations. arXiv:2501.16495 [cs.SE] https://arxiv.org/abs/2501.16495