Judging an LLM on judging LLM-generated Microservice Documentation: A Supplemental Study and Observations

Dawn Mai, 25R50028 | Supervised by Takashi Kobayashi, School of Computing

Abstract

This study examines the reliability of an LLM-as-a-Judge framework used to evaluate LLM-generated microservice documentation within a Graph RAG–based system. After identifying multiple issues in the original framework, which included KG retrieval errors, hallucinated service descriptions, and inconsistent scoring, we applied iterative debugging and prompt refinement. Testing across seven repositories revealed that although these fixes improved stability, judge accuracy declined as repository chunk size increased. RAG-based documentation often matched or exceeded KG/GNN outputs for larger repositories, suggesting that evaluation quality remains sensitive to prompt coverage and system complexity.

1 Introduction

1.1 Background

Microservice architecture is a software design which runs on the cohesion of many smaller, independent computing processes (microservices). [1] This approach allows for speedier development and a more efficient deployment process, but consequently, more microservices mean increased maintenance and effort in their documentation for the sake of reliability. [2,3] Given the rise of LLMs and their use cases in automating the software development workflow, it is no surprise that their use has been extended to code comprehension for software documentation generation purposes. Though RAG has shown to be a promising approach [4], it struggles to comprehend large, complex codebases and long context windows without additional knowledge on the structure of these inputs. This structure is what's needed to generate standard microservice architecture documentation.In the Software Analytics Research Group at the School of Computing, a senior has proposed a method to generate standard microservice architecture documentation by using LLM as follows: [5]

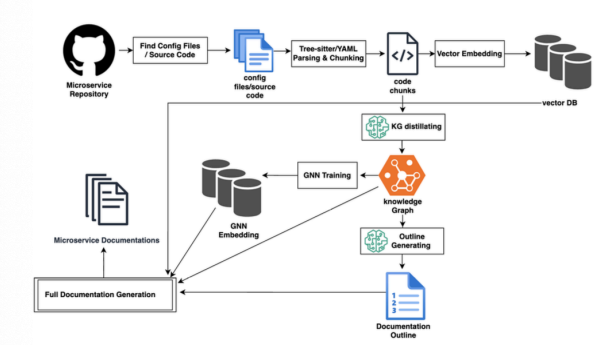


Figure 1: *System Architecture Overview [5]*

1. Parse and split those files while injecting metadata as necessary with existing popular tools. Extracted chunks are stored in a FAISS vector database.
2. Perform a Knowledge Graph (KG) distillation on all chunks using Large Language Model (LLM) extraction and prompting strategies. Distilled KG is stored in a graph database (Neo4j)
3. Train a Graph Neural Network (GNN) from the KG to get node embeddings and generate a documentation outline with LLM using the KG as context.
4. Generate the full documentation section by section using the outline as the blueprint, the KG, GNN, and vector DB as context retrieval sources.

They gave us various microservice architecture repositories to execute the framework code (see Figure 1 for the overview of the method). [5] At the end of program execution, a different LLM model acting as a judge would be executed on each repository's generated documentation and provide numbered scores to each document based on a "universal architect" prompt.

1.2 Motivation

Upon executing the original framework code with the provided repositories [5], we encountered several bugs and issues, including but not limited to:

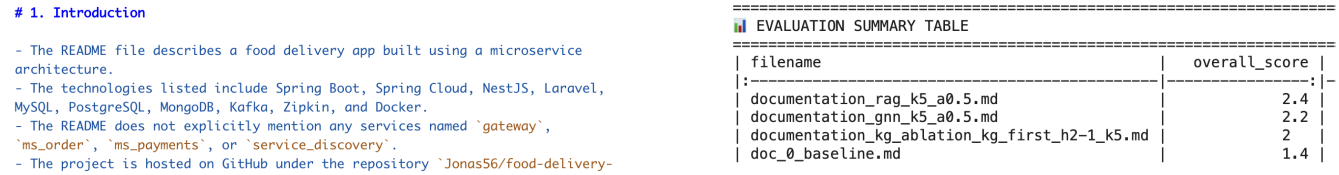
```
# 1. Introduction

- The README file describes a food delivery app built using a microservice
architecture.
- The technologies listed include Spring Boot, Spring Cloud, NestJS, Laravel,
MySQL, PostgreSQL, MongoDB, Kafka, Zipkin, and Docker.
- The README does not explicitly mention any services named `gateway`,
`ms_order`, `ms_payments`, or `service_discovery`.
- The project is hosted on GitHub under the repository `Jonas56/food-delivery-
app`.
- The app can be run using Docker, with instructions provided for using
`docker-compose`.
- Contributions follow a "fork-and-pull" Git workflow.
- The project is distributed under the MIT License.

# 2. High-Level Architecture

*No info found (Graph Disconnected).*

# 3. Service Details

## `gateway`

*No info found (Graph Disconnected).*

*No info found (Graph Disconnected).*

*No info found (Graph Disconnected).*
```

Figure 2: *Screenshot of an example of "Graph Disconnected" error message in the KG-based LLM-generated document.*

1. TypeErrors and "Chunk file not found" messages.
2. "No info found (Graph Disconnected)" error messages in the KG-ablation document, even if the KG was fully constructed. See Figure 2.
3. Low scores (scores of less than 3) given to KG- and GNN-based LLM-generated documents. The theory in [5] states that these two should be rated higher than all other documents due to their providing a structure to the LLM on which to generate documents. See Figure 3.
4. RAG-based LLM-generated documents scoring higher than KG- and GNN-based documents. This was more obvious with increases in chunk size for some repositories. Figure 3 also demonstrates this.

1.3 Research Goal

Further investigation was needed, as such issues were not found on our senior's execution of the same code on the same repositories on their machine. With permission to edit the original framework code, we set out to correct these issues with the help of AI tools (e.g. Cursor) and manual debugging. We intended to have the code correctly evaluate and rank the generated documentation for a new, partially-derived dataset from the original repositories tested in [5], with the hypothesized ranking as follows:



```
================================================================
📊 EVALUATION SUMMARY TABLE
================================================================
| filename                                          |   overall_score |
|:--------------------------------------------------|----------------:|-
| documentation_rag_k5_a0.5.md                      |             2.4 |
| documentation_gnn_k5_a0.5.md                      |             2.2 |
| documentation_kg_ablation_kg_first_h2-1_k5.md     |             2   |
| doc_0_baseline.md                                 |             1.4 |
```

Figure 3: *Screenshot of initial scores given to [Nasruddin/spring-boot-based-microservices] documents with the original framework code.*

- KG/GNN documents (at least a score of 3)
- RAG documentation (a score of 1-3)
- Baseline documentation (a score below 2)

If the amended code still could not do this, we also intended to perform further hypothesis testing and factor analysis to determine the root cause.

2 Method

2.1 Dataset

Seven repositories were chosen in total, with chunk sizes ranging from under 100 to above 500, but no larger than 1000. Of the original repositories tested [5], only three were selected to observe differences in between the results of the original framework and the edited one for this study (marked with an asterisk):

- LauroSilveira/microservices-java-spring-boot*
- Jonas56/food-delivery-app*
- C0lliNN/EventDrivenCommerce*
- Nasruddin/spring-boot-based-microservices
- rcherara/microservice-architecture
- ocp-power-demos/sock-shop-demo
- cer/event-sourcing-examples

Aside from being popular, open-sourced microservice architecture software repositories with architecture diagrams, all of them were chosen due to their small/medium chunk size, indicating both complexity and a quick analysis runtime [5].

2.2 Iterative code improvement

We started and edited the framework code with Nasruddin/spring-boot-based-microservices due to its relatively low chunk size of 199, indicating a complex microservice architecture with a quick enough documentation generation time. Whenever an error or ranking issue was observed, the LLM-as-a-judge code was then executed a few more times to ensure the

```
============================================================
📊 EVALUATION SUMMARY TABLE
============================================================
```

filename	overall_score
documentation_gnn_k5_a0.5.md	4.6
documentation_kg_ablation_kg_first_h2-1_k5.md	4.4
documentation_rag_k5_a0.5.md	4.2
doc_0_baseline.md	1.2

Figure 4: *Screenshot of the final scores for [Nasruddin/spring-boot-based-microservices] documents after framework code was fixed for issues.*

error was consistent before being debugged and addressed with the help of AI tools. After several iterations, we were able to address the issues listed in Section 1.2 above with the reasons and fixes for them as follows:

1. TypeErrors due to missing imported packages: all packages needed were now installed at the beginning of the program.
2. "No info found (Graph Disconnected)" error messages in the KG-ablation document: this was fixed by forcing the framework to use a semantic-based fallback and to locate the graph rather than an empty list.
3. Low scores (less than 3) given to KG- and GNN-based LLM-generated documents: this was in part due to LLM hallucination of the presence/absence of services, which can result in documents stating that a certain microservice was/wasn't present when the opposite was the case. This was fixed by tightening the “universal architect” prompt to penalize such contradictions, with a lighter penalty if the document in question still supplied information to answer the LLM-as-a-judge. In addition, the “universal architect” prompts were edited to use stricter, specific criterion, since we were looking for a certain ranking.
4. RAG-based LLM-generated documents scoring higher than KG- and GNN-based documents: the fixes in 3. were also applied here. No manual boosts were used to interfere with the data, and neither the code generating these documents was altered; scoring only used the material of the documents provided and the repository they reported on.

2.3 Document generation and judging

After these issues and bugs were fixed and the code

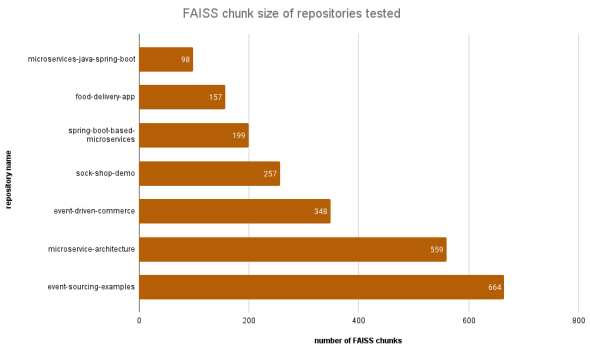


Figure 5: *Chart displaying the sizes of the repositories tested in FAISS chunks. For a larger view, click on the link in the appendix.*

correctly generated and scored documentation for [Nasruddin/spring-boot-based-microservices], were approved by our senior, we could then use the code on the other six listed repositories by changing the GitHub links and names of the local repository paths used. The whole notebook was executed exactly once per repository, as we only had one Neo4j instance to store microservice data on. LLM-as-a-judge code was then executed six times per repository to compute an average score for the documents (including the initial, whole-notebook execution). All repositories were analyzed one at a time. See Appendix A.1 for the full results.

3 Results

With more chunks, there are more microservices to document as well as a longer runtime to generate and grade documentation (see Figure 5). We observed that the LLM did a better job of generating accurate documentation and giving such documentation appropriate, higher scores with repositories that contained *fewer* chunks (see Figure 6). The more chunks a repository had, the higher the scoring for RAG documentation. In those cases, RAG documentation was either just below or surpassed average for both KG and GNN documentation. This is highly unusual, as we hypothesized RAG to be given lower scores with bigger repositories due to the amount of information that must be sifted through (think “crowded vector spaces” where information is tightly packed). Given this is not the anticipated ranking order due to the original thesis [5], we suspect:

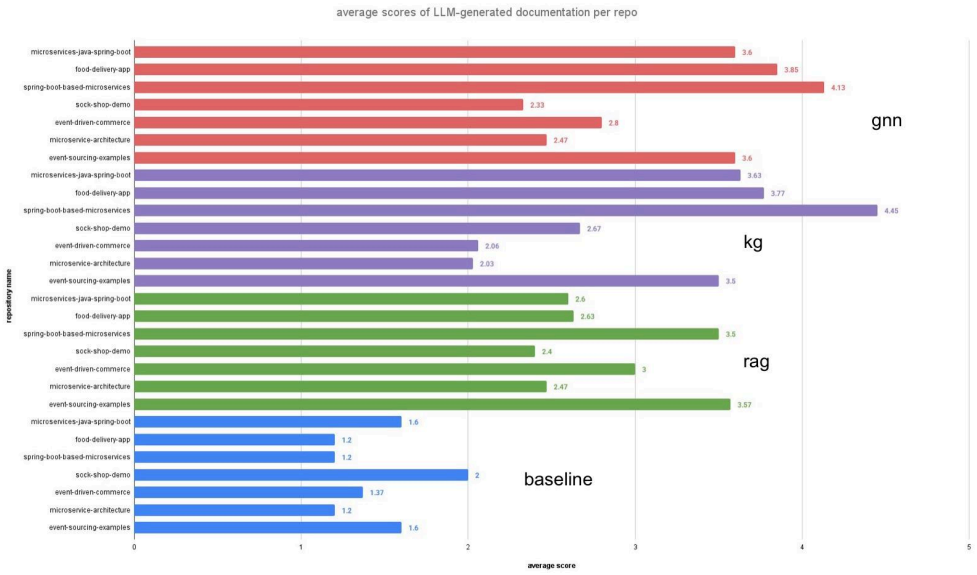


Figure 6: *Chart displaying average scores per repository, color-coded by document type. For a larger view, click on the link in the appendix.*

1. The "universal architect" prompt we used still did *not* cover enough potential services that a given microservice architecture would use, or
2. Our methods for generating KG, GNN, and RAG documentation have been producing inconsistent results relative to the repository being graded.

4 Conclusion

In this study, we improved upon a pre-existing code framework to evaluate LLM-generated microservice documentation in the interest of fixing various issues and bugs when such code was initially executed. In the process, we observed an interesting pattern relating FAISS chunk size to RAG-specific documentation in addition to the overall trend that with more code chunks, LLM-as-a-judge gives out lower, less accurate scores to LLM-generated microservices documentation.

4.1 Limitations

There are several limitations to this study, most notably the cost per OpenAI LLM call and the number of repositories we could conduct the study on in the given timeframe. Since we were only allowed one instance to write KG data to, we were unable to test our framework on massive industrial-scale repositories with hundreds of services and highly complicated topologies. LLM-as-a-judge may still exercise preference, and since they are nondeterministic by nature, randomness in the output can occur, which may require an adjustment of the prompt or certain sections of code to better accommodate the repository being judged, since we prioritized diversity.

5 References

[1] N. Dragoni et al., "Microservices: yesterday, today, and tomorrow," 2017.

[2] J. Bogner, et al., "Assuring the evolvability of microservices: Insights into industry practices and challenges," Sept. 2019.

[3] J. Bogner et al., "Microservices in industry: Insights into technologies, characteristics, and software quality," in 2019 IEEE International Conference on Software Architecture Companion (ICSA-C), pp. 187–195, 2019.

[4] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive nlp tasks," 2021.

[5] L. Qiao, "Structure-Aware Enhanced LLMs via Knowledge Graphs for Microservice Architecture Documentation", 2026. Master's Thesis, School of Computing, Institute of Science Tokyo.

6 Appendix

A.1 Google Sheet of raw data (linked)