Grading an LLM on grading LLM-generated Microservice Documentation: A Supplemental Study and Observations

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

Abstract

The usage of Large Language Models (LLMs) to generate documentation for microservices architecture repositories has become more common as the number of microservices in a given software system grows, making it more difficult for users and contributors to quickly understand overall system behavior. This research focuses on using the LLM-as-a-judge framework in ranking and assessing the quality of LLM-generated documentation—namely Retrieval-Augmented Generation (RAG), our own Graph Neural Network- (GNN) and Knowledge Graph-based (KG) frameworks, and a contextless baseline for each microservices repository studied—to observe whether or not such documentation can be generated accurately even if given proper context via GNNs and KGs, and where the gap lies in providing semantically and structurally accurate content.

1 Introduction

1.1 Background

Microservice architecture is an application 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 means increased maintenance of said microservices and more 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 in the matter [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.

1.2 Research Goal

The goal of this research is to provide said structure to a given microservices architecture repository via constructing a KG and then a GNN based on said KG. By giving this additional context to the LLM and explicitly modeling the latent dependencies between microservice components and services, we hope to enable the LLM to retrieve deeper context and therefore be able to generate high-quality documentation.

2 Methods

2.1 Overview

For the sake of brevity, the fully proposed novel framework is shown in Figure 1 below, with the following explanation of the system:

1. The system starts with taking a microservice repository URL (web) as input and clones it to the local environment (a Google CoLab notebook), then applies pre-defined heuristic based filters to extract the important configuration files and source code files.

2. We leverage existing popular tools to parse and chunk those files while injecting metadata as necessary. FAISS chunks are stored into a vector database (known herein as "chunks").

3. We perform a KG distillation on all chunks using LLM extraction and prompting strategies.

4. Once the KG is constructed and stored into a graph database (we use an instance of Neo4j here), we train a GNN from the KG to get node embeddings and also to generate a documentation outline using the KG as context.

5. Finally, we generate the full documentation using the outline as the blueprint, using the KG, GNN and vector database as context retrieval sources.

This paper focuses on the last part: full documentation generation and the resulting microservice documentation.

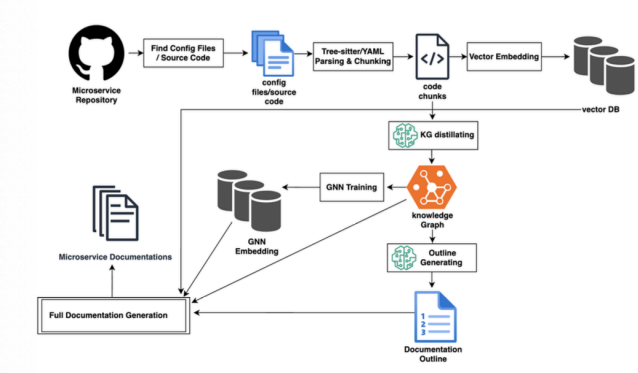

Figure 1: System Architecture Overview

2.2 Repository verification

The repositories were sourced off of GitHub and had to meet the following requirements:

1. They must be open source, i.e. accessible to the public. We prioritized repositories with more than 10 GitHub stars. In GitHub, a star indicates that someone has "favorited" a repository, either for contribution or use of the software code it houses. The more stars a repository has, the more "popular" it is. Using this logic, we wanted to focus on "busy" repositories, those with a lot of internet traffic and thus a greater range of code diversity through contributions.

2. Have an architecture diagram provided by maintainers/authors. The diagram is usually included in the README to demonstrate to readers how the microservices for the reference of ground truth microservice architecture. An example of such a diagram is shown below in Figure 2.

3. Covers diverse architectural patterns with keywords such as pub/sub, REST (HTTP), and Event-Driven, as well as utilizing a popular framework such as Spring Boot.

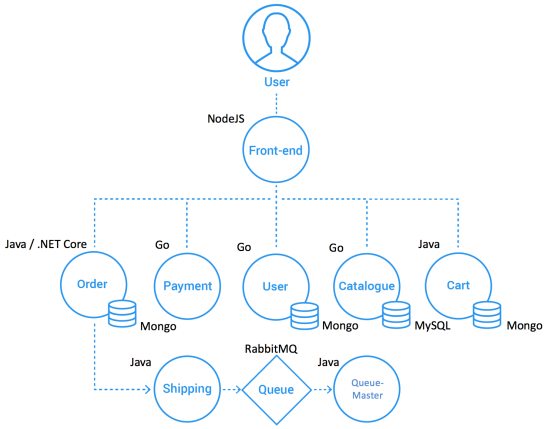

Figure 2: Architecture diagram of ocp-power-demos/sock-shop-demo.

Manual search as well as the use of AI tools (e.g. Cursor) were used to verify suitable repositories. The repositories we chose are:

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

2.3 Repository analysis

Each repository operated on its own local environment and instance of the proposed framework, though they all relied on one Neo4j instance for their KG. Therefore, code to build the KG for each repository was run exactly once and repositories were analyzed one at a time. Logs detailed how many FAISS chunks a repository consisted of; a repository was only considered suitable for the study if it was less than 800 chunks in the interest of time.

2.4 Documentation generation

Once the KG is built, code to build the resulting GNN and relative documentation is then executed. Each repository has the following Markdown (.md) files

generated by GPT-4o and judged by GPT-5.1 on a scale of 1 to 5:

- *documentation_gnn_k5_a0.5.md* (repository documentation based on the graph neural network, sourced from the knowledge graph)
- *documentation_kg_ablation_kg_first_h2-1_k5.md* (repository documentation based on the knowledge graph, sourced from Neo4j)
- *documentation_rag_k5_a0.5.md* (repository documentation based only on RAG)
- *doc_0_baseline.md* (repository documentation based on plain LLM prompt and output with no further context (i.e. GNN and KG not included in the prompt))

Based on our proposed framework, we expect the GNN and KG documents to rank higher than the RAG and baseline documents on average, as they reflect the structure of the microservices in a given repository more accurately through the explicit displaying of the relationships between them. The average scores of the generated documentation was obtained by executing only the CoLab cell containing the LLM prompt about 6 times; documentation of each attempt per repository was recorded on Google Sheets, which can be found here. The judge was given the following prompt in Figure 3 below, which is repository-agnostic:

```
JUDGE_SYSTEM_PROMPT = """You are a Principal Software Architect acting as a Judge.
Compare the [GENERATED DOCUMENTATION] against the [GROUND TRUTH REPO].

**ABSOLUTE RULE:** Use ONLY the provided ground-truth context (file structure, configs, service list).
Do NOT rely on prior knowledge of any repo or domain. If a claim is not supported by the ground-truth
context, treat it as hallucination.

**EVIDENCE STANDARD:** Prefer statements that can be directly verified from the provided context.
Penalize unsupported specificity more than cautious generality.
Penalize internal contradictions (e.g., claiming "no mention of X" while also describing X).

**SERVICE CONSISTENCY RULE:** The provided service list is authoritative. Mentions of services
not in that list should reduce **faithfulness** and **correctness**. Missing most services
should reduce **completeness**.

**GENERALITY RULE:** The rubric must apply to ANY repository. Do not assume microservices unless
supported by ground-truth evidence.

### **GLOBAL METRICS (1-5)**

**(1) COMPLETENESS**
* **5:** Covers ALL applicable sections with high detail from evidence.
* **1:** Misses major functional blocks that are clearly present.

**(2) CORRECTNESS**
* **5:** Details match ground-truth exactly.
* **1:** Hallucinates dependencies, ports, services, or versions.

**(3) FAITHFULNESS**
* **5:** Claims are grounded in context.
* **1:** Invents functionality or entities not supported by context.

**(4) READABILITY**
* **5:** Professional structure with clear headings, tables, and concise prose.
* **3:** Understandable but inconsistently structured.
* **1:** Unstructured or hard to follow.

**(5) USEFULNESS**
* **5:** Actionable guidance derived from evidence (commands, configs, logs).
* **3:** Descriptive but not operational.
* **1:** Vague or marketing-like.
```

Figure 3: Prompt for LLM-as-a-judge

3 Results

Figure 4 below shows how "big" each repository is in terms of the amount of chunks processed to build the respective knowledge graph and graph neural network for it. It follows that with more chunks, there are more microservices to document as well as a longer runtime for our novel framework to generate and grade documentation using the repository itself as a sort of "gold standard".

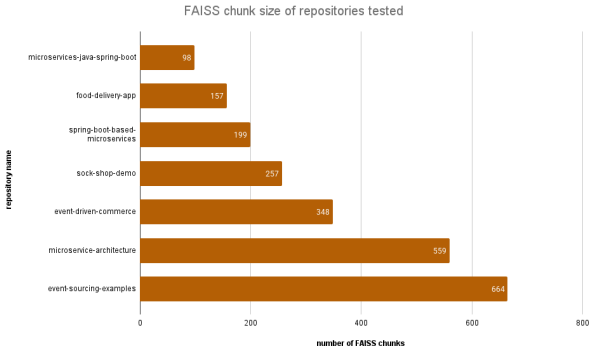

Figure 4: Chart displaying the sizes of the repositories tested in FAISS chunks. For a larger view, click on the chart to be redirected.

Overall, the LLM did a better job of generating accurate documentation and giving such documentation appropriate, higher scores with repositories that contained less chunks. The more chunks a repository had, the higher the scoring for RAG documentation. (Figure 5 below demonstrates this in more detail.) In those cases, RAG documentation was either just below or surpassed average for both KG and GNN documentation.

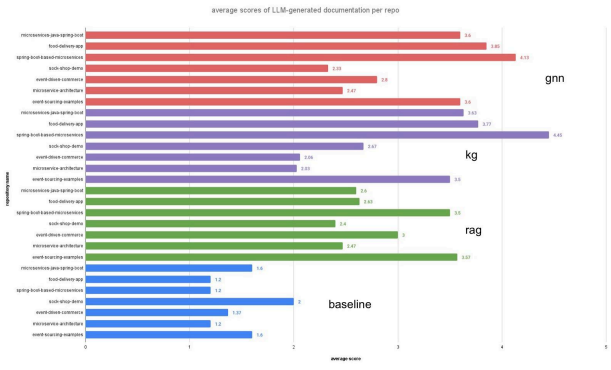

Figure 5: Chart displaying average scores per repository, color-coded by document type. For a larger view, click on the chart to be redirected.

Given that this is *not* the anticipated ranking order due to both KG and GNN providing the "structure" for the LLM to better grade such documentation, we suspect:

1. The prompt we used 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.

The occasional bug in the framework code was encountered during experiment runs and was fixed with the help of AI tools (e.g. Cursor), but even in spite of this and tightening the judgment criteria, this odd pattern persisted. Given this, we recommend additional work in prompting techniques and how they influence the efficacy of LLM-as-a-judge for documentation.

4 Conclusion

4.1 Overview

In this study, we proposed a novel Graph-enhanced structure-aware Retrieval Augmented Generation (RAG) framework that integrates Knowledge Graphs (KG) with Graph Neural Networks (GNN). We also witnessed 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.

4.2 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 on the free plan of Neo4j, we were only allowed one instance to write KG data to and 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. Nevertheless, these risks were minimized through strict prompting, calculating the average scores, and manual verification of the results.

4.3 Future Work

Due to limited time and computational resources, we keep the following tasks as our future works:

1. Validate the proposed method on more massive larger scale repositories to assess its performance beyond the selected repositories' scale used in this study.

2. Improve the way the ground truth microservice architecture topology is extracted from a target repository, moving beyond the limitation of requiring the target repository to provide an architecture diagram.

5 References

[1] N. Dragoni, S. Giallorenzo, A. L. Lafuente, M. Mazzara, F. Montesi, R. Mustafin, and L. Safina, "Microservices: yesterday, today, and tomorrow," 2017.

[2] J. Bogner, J. Fritzsch, S. Wagner, and A. Zimmermann, "Assuring the evolvability of microservices: Insights into industry practices and challenges," Sept. 2019.

[3] J. Bogner, J. Fritzsch, S. Wagner, and A. Zimmermann, "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, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W. tau Yih, T. Rocktäschel, S. Riedel, and D. Kiela, "Retrieval-augmented generation for knowledge-intensive nlp tasks," 2021.