2024 IEEE International Students' Conference on Electrical, Electronics and Computer Science

2024 IEEE International Students' Conference on Electrical, Electronics and Computer Science (SCEECS) | 979-8-3503-4846-0/24/$31.00 ©2024 IEEE | DOI: 10.1109/SCEECS61402.2024.10482119

Automated API Docs Generator using Generative AI

Prakhar Dhyani
Department of Computer Science and Engineering
Graphic Era Hill University
Dehradun, India
prakhardhyani1000@gmail.com

Shubhang Nautiyal
Department of Computer Science and Engineering
Graphic Era Hill University
Dehradun, India
Shubhang999454@gmail.com

Aditya Negi
Department of Computer Science and Engineering
Graphic Era Hill University
Dehradun, India
Negiaditya1234@gmail.com

Shikhar Dhyani
Department of Computer Science and Engineering
Graphic Era Hill University
Dehradun, India
Shikhardhyani025@gmail.com

Mrs. Preeti Chaudhary
Department of Computer Science and Engineering
Graphic Era Hill University
Dehradun, India
Priti.chaudhary1989@gmail.com

***Abstract*— Our study provides an improvement on the creation of Application Programming Interfaces (APIs) usage documentation using the efficiency and power of Generative AI. APIs play an important role in software integration and software maintenance but the process of API documentation creation has been traditional and did not evolve with time, this paper employs Generative AI to enhance the accuracy, speed, and scale of API documentation generation. The automated API documentation generator is created using natural language processing applied through a large language model (TinyPixel/Llama-2-7B-bf16-sharded model). Training data was created by applying web scraping on various large tech companies' documentation web pages to get a good quality and industry-standard documentation dataset. It was further diversified and increased using the GPT model to handle a wide range of API scenarios. The fine-tuning greatly enhanced the TinyPixel/Llama-2-7B-bf16-sharded model's efficiency and quality of output which is proven by the reduced response time and the accuracy of documentation generated. Our study's comparative study confirms the effectiveness of the approach used. Our study's conclusion offers a comprehensive approach that should improve software development processes and pave the way for additional developments in API documentation.**

Keywords*— *API Documentation, Generative AI, Fine-Tuning, Large Language Models, Web Scraping, Natural Language Processing.

I. Introduction

APIs (Application Programming Interfaces) allow different software components to easily communicate with each other. To provide usability, integration and to maintain complex architecture, comprehensive documentation is required. Earlier the manual or partially automated creation and maintenance made it difficult to keep up with the ongoing rapid development updates. Therefore, an AI driven API documentation generator is a better approach for creation of API documentation which we suggested in the paper.

This system is based on Generative AI technology that combines machine learning and natural language processing to provide tools that are capable of handling difficult task that mainly require human input. This strategy might make a bigger impact on how API Documentation can be written. The goal of this research is: (1) to evaluate the economic and practical advantage of incorporating AI technology. and (2) to show how AI technology improves data driven operations by giving them access to the speed and efficiency in modern software development. The intended goal is to guarantee comprehensive, precis and up to date API docs. Examining the state of API strategy documentation at the moment and highlighting the limitations and flaws of the methods used is the goal of this study. To address these issues, the paper proposes generative AI, which gives readers access to a tool that might greatly increase the precision, speed, and scalability of API documentation creation.[1]

II. Background and Evolution of Application Programming Interfaces

APIs, or application programming interfaces, are crucial in the large field of software development because they allow the linkage and integration of different applications. To understand the importance of APIs, one must first explore

[1] API - Wikipedia

their origins, history, and critical functions in today's software ecosystem. As software development progresses, new techniques emerge, each with its own set of advantages and uses. API documentation that is clear and thorough is in high demand. Among them are:

A. Manual Documentation

The conventional method of producing API documentation is having a developer or author write a book, usually with the use of a text editor or platform like Microsoft Word or Confluence. This method takes a lot of time and work, but it allows for extension and customization. One of the main problems with this approach is keeping the database compatible with constant API updates.[2]

B. Automated Generation from Code Comments

In order to extract relevant information, programs like Javadoc for Java and Doxygen for C++ take instructions straight out of the source code. Consistency between API usage and related documentation is supported by this integration. Nevertheless, the comprehension and caliber of the code's comments determine how successful this strategy will be.

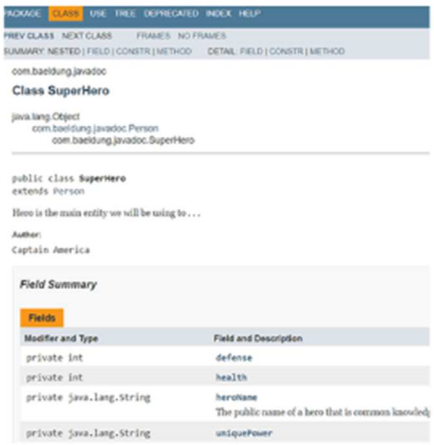

Figure 1 Sample Javadoc Documentation for a Java Class [3]

C. API Description Languages

Languages such as OpenAPI (previously known as Swagger) and RAML provide developers with the means to describe APIs in machine-readable formats. These descriptions act as a base for generating structured documentation, which aids in maintaining uniformity and simplifies the process of updating the documentation. This approach enhances the efficiency and consistency of API documentation.

D. Framework-Specific Documentation Generators

Some web frameworks come with integrated or additional tools specifically for generating documentation. Examples include the Django Rest Framework for Python and Spring Rest Docs for Java. These tools enable the creation of documentation directly from the framework's ecosystem, ensuring a high level of coherence with the API's development environment. This integration, which is closely linked with the entire development, streamlines the information process.

E. Interactive API Documentation

Interactive interfaces by Postman and Swagger UI sets apart them from standard manuals. Users can search and make API queries straight from within the file. This helps in testing, flight testing and improves of API Documentation.[4]

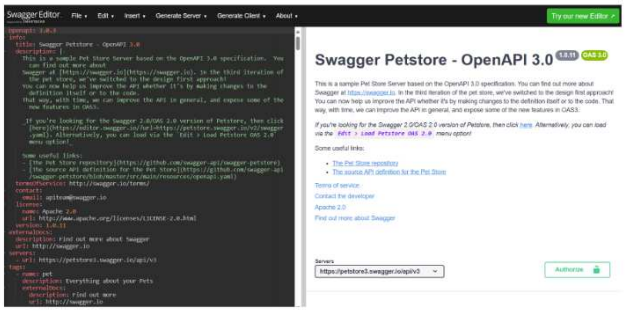

Figure 2 Example of Interactive API Documentation using Swagger UI [5]

F. Version Control and Synchronization Tools

Version control systems used in Read Documentation platforms let you update both the code and its documentation together. These systems automatically refresh the documentation when the code changes, making the information more accurate and reducing the amount of data that needs to be managed to keep everything current.

G. Generative AI and Machine Learning

Using advanced algorithms and machine learning, systems can automatically detect patterns, update code, and manage large volumes of data on their own. This represents a big step forward in how software documentation is created and maintained.

In conclusion, we're shifting our attention to using AI to make API documentation better and more up to date. This means that documentation will be able to keep up with the fast changes in software development, making everything more user-friendly and adaptable.

III. Literview Review

Reference [1] presents a way of improving user experience in producing interactive documentation for Web APIs. In this method, computational approaches are employed to generate and interpret natural language descriptions that are useful to automated systems as well as human users. The study was conducted with data-related domain enthusiasts to validate the efficacy of this methodology. It has been discovered that it can significantly reduce time and effort required for understanding and using web APIs thereby demonstrating its utility in API description. The main purpose of [3] is to investigate how scaling up computational language models influence their performance on tasks with fast learning rates. The research thoroughly compares various model sizes and configurations. This study found that making the models larger improves the speed of learning by a great extent through some experiments made with different model scales and little data for fine-tuning purposes. These enlarged models perform on par with or even

[2] API - Wikipedia
[3] https://www.baeldung.com/javadoc
[4] Intro to APIs: History of APIs | Postman Blog
[5] https://editor.swagger.io/

better than previous models that relied on large amounts of data.

RbG has been introduced as a documentation tool specifically designed for scientific and engineering software [6]. In generating documentation, this program automatically extracts mathematical formulas and decision-making logic from the code through code analysis. RbG arranges these documents with comments within the source code itself giving it total control over the content that is provided. RbG's usage in many different professional contexts such as reverse engineering, documenting new software projects and updating existing system documentation demonstrates its flexibility. These case studies demonstrate how RbG can be customised to satisfy different needs related to software development documentation.

Reference [7] looks on the benefits of combining more general, more generic instructions with specialist models for natural language task processing. This study compares models trained on general data with those designed for specific tasks to investigate the effects of adding extra data on performance. The results are quite impressive when training data is scarce. The main finding of this work is that reliable, high-quality generalist data are essential to prevent models developed for tasks from performing inadequately. This work highlights the difficulties and complexities of carefully integrating data to enhance natural language processing algorithms.

Table 1 Comparative Analysis of Large Language Model Applications and Methodologies

Reference	Purpose	Model/Technique Used	Dataset Used	Comparison Parameters and Methodologies	Results/Findings
[2]	News summary generation using LLM	LLM with evolutionary fine-tuning.	Niche domain dataset, PENS, grain storage pest	Compared with TFIDF and TextRank algorithms	**NSG generates accurate, reliable summaries**
[4]	Adapting language models to society	GPT-3 language model	Hand-curated dataset with target values.	Metrics: output adherence, toxicity, common word	Metrics for PALMS process evaluation
[5]	CodeBERT for programming and NLP	CodeBERT pre-trained model	Provided by Husain et al. (2019)	Pre-trained on large-scale corpus, fine-tuned on NL-PL apps	Performance on code documentation, retrieval
[9]	Compute-optimal training of transformer	Chinchilla (Transformer language model)	Not explicitly mentioned	Investigated optimal model size and tokens	Compute-optimal training solution
[10]	BERT fine-tuning for text classification	BERT (Bidirectional Encoder Representations)	IMDb, Yelp P., Yelp F., TREC, Yahoo! Answers, AG's News, DBPedia, Sogou News	Investigated various BERT fine-tuning methods	Achieved new state-of-the-art results on text classification datasets
[13]	Fine-tuning pre-trained language models	RoBERTa LARGE, mBART LARGE	GLUE benchmark	Two-stage fine-tuning approach	Task-agnostic mask, adapter fine-tuning

Reference [8] investigates the tranGAN model about the use of Generative Adversarial Networks (GANs) for text generation. This model combines the actor-critic technique with transformer architecture to address common text production issues including sequence dependence and exposure bias. The Penn Treebank dataset was used to assess tranGAN's capacity to generate grammatically sound and logical sentences. These assessments show off tranGAN's text creation skills.

A prompt tuning technique is presented in [11] that makes it possible to condition language models that have already been trained for specific tasks in an effective manner. The method works effectively on a variety of natural language processing applications and requires less training time and parameters than other traditional fine-tuning techniques. To make a model produce targeted outputs that does not require a lot of additional training, prompt tuning allows for the customizing of a prompt vector for that task. Tasks such as question answering, natural language inference, and text categorization show how well this technique works, not only making it much more robust but also allowing combining efficiently and quickly thereby enhancing its general applicability.

The purpose of [12] is to create greater transparency in evaluating language models. It provides a comprehensive framework called HELM that evaluates these models on various aspects such as toxicity, efficiency, bias, fairness, robustness, accuracy, and calibration. The evaluation of 30 different popular language models on 42 scenarios makes

HELM extend the scope of language model evaluation greatly. Moreover, it outlines the trade-offs between metrics and models while also providing a benchmark to compare different generative AI techniques across languages. A detailed examination of generative AI can be found in [14], which discusses several advanced computational techniques for creating meaningful content. Generative AI is a rapidly expanding field which must consider the limitations as well as opportunities therein. The methods involve presentation of variational autoencoders (VAEs), generative adversarial networks (GANs) and deep learning with examples on how they are used in different types of content creation such as writing, music and photos. This paper examines the problems facing the development and deployment of Generative AI systems to name a few large training data set requirements, worries about potential biases and moral dilemmas.

IV. Proposed Methodology

The project aims at introducing artificial intelligence in API documentation. However, the specific objective is to develop an AI model that can instantly generate accurate documentation for various APIs. This comprehensive approach will include creating a diverse dataset, fine tuning a massive language model and designing a user-friendly interface for inputting API details. Ultimately, it should improve the accuracy and efficiency of API documentation hence maintaining effective software development process flow. The procedure is divided into several steps:

A. Dataset building and preparation

This step involved the collection of data from web-scrapping on various top leading companies of the world. We collected the open-source API documentation provided by these companies to have a clear goal of achieving best documentation possible. Some of the companies were – PayPal, Google Map Apis, Stripe, etc.

The data collected was further shaped into input and output objects to have a concise and clear understanding of our model to work on. To diversify and enlarge the data, artificial data was generated through GPT. This was a critical step to diversify the dataset to tackle all the API edge test cases. Lastly, the dataset was formatted according to standards for optimizing AI models for NLP tasks, such as modeling human – assistant interaction.

B. Fine Tuning Process

To get the most out of AI capabilities in this project, some changes had to be made on LLaMA 2- 7B parameter sharded model. This model was necessitated because it has rich language generative and understanding capacities necessary for generating accurate API docs. Fine-tuning process became easier with AutoTrain package that provided frameworks and tools able to efficiently optimize different training settings. In this step, therefore, we had to make sure that the model was trained adequately enough to understand API documentation while concentrating much on generating highly relevant outputs that were very accurate at the same time.

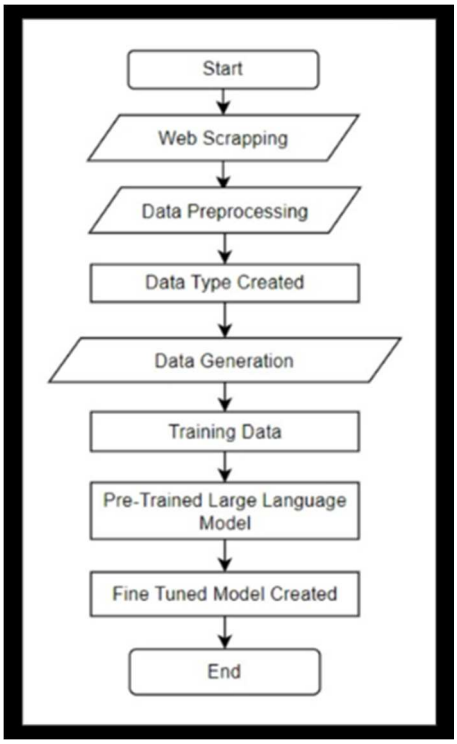

Figure 3 Workflow Diagram for AI-Driven API Documentation System

C. Generation and Evaluation

The deployment of the model marked a significant advancement in the generation and evaluation of API documentation. For the model to give better responses, we improved it by using GenerationConfig. This helped us adjust important settings to get the best performance. We made sure the model was doing a good job by checking how fast it works and how good is the created API documentation.

D. Output

After the model was evaluated, we added a user-friendly interface using streamlit library for the user to provide API details. The model then creates API documentation in JSON format. The output in JSON format is then displayed in a readable and interactive manner using HTML, CSS, and JS. The API docs given by our model is well represented in form of an interactive webpage.

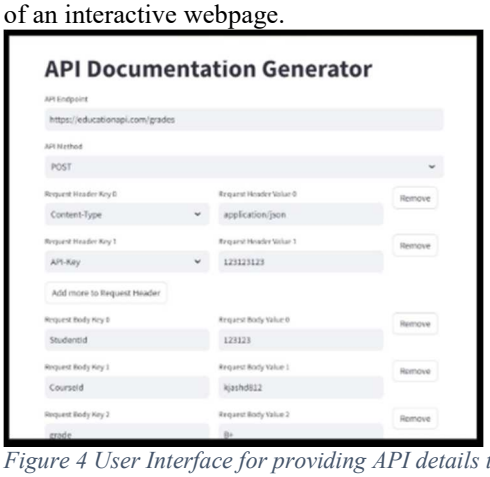

Figure 4 User Interface for providing API details to the model

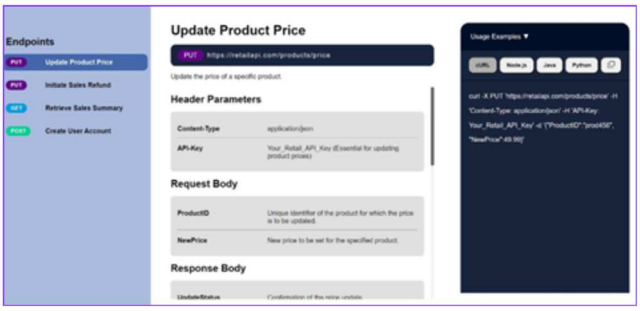

Figure 5 Sample Output from our API Documentation

V. Result Analysis

Our focus was to maximize the performance of the TinyPixel/Llama-2-7B-bf16-sharded model using a dataset meant for API documentation to fine tune the model. Our finetuned model outperformed the baseline model in several categories.

A. Initial Model Performance

There were problems in the initial responses by TinyPixel/Llama-2-7Bbf16-sharded model. To produce API documentation, it took an average of fifty seconds. Furthermore, these outputs frequently fell short of the assignment's true requirements in terms of the accuracy and comprehensiveness needed to create an excellent API specification. The first findings show that the model misinterpreted the user's request for API documentation.

B. Performance after Refinement

After the model was adjusted, both its output quality and speed significantly increased. These days, it only takes 36 seconds on average to construct an instance of API documentation. This acceleration is necessary for practical applications, especially in settings where software development proceeds rapidly. Furthermore, there was a discernible improvement in the quality of the documents generated. The updated model adhered to professional API documentation standards and produced documentation that was more precise, intelligible, and appropriate for the current scenario.

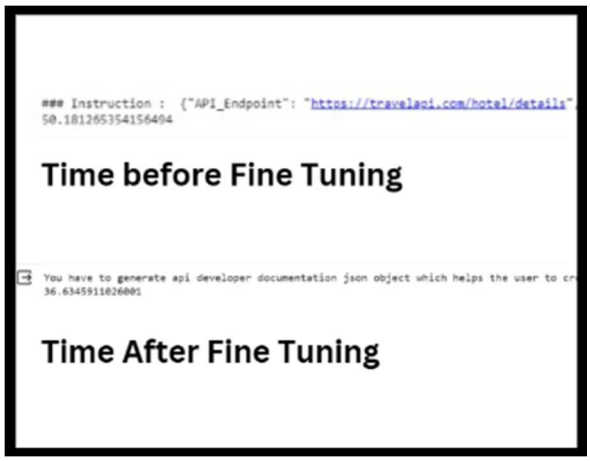

Figure 6 Performance Comparison Before and After Model Fine-Tuning

C. Comparative Analysis

A comparison of the fine-tuned and original performances reveals a large difference. In terms of generation speed as well as content correctness and relevancy, the upgraded model performs better than the original. This illustrates the usefulness of our approach and the significance of using a dataset that has been specifically created for a particular task. The results from our dataset was not only appropriate and useful but also took less time to get generated, after the model had a good idea of what to generate.

Figure 7 Comparison of API Documentation Outputs Before and After Fine-Tuning.

VI. Conclusion

The process of creating and maintaining API documentation for modern applications has been made easy through this our study, which leverages the power of Generative AI to create API documentation for user which is concise and meets the industry standards. We optimized the TinyPixel/Llama-2-7Bbf16-sharded model to achieve significant speed and quality gains. The improvements shown by the model show its flexibility and accuracy. This expands the potential of machine learning techniques. Using web technologies to interpret the model's output, we enhance user experience and provide a visually appealing and captivating interface. Developers and other stakeholders will find it simpler to understand and learn about API integration as a result. The addition of HTML, CSS, and JavaScript has made the API documentation easier to read and comprehend. Now that documentation can be accessed more quickly and perceptively, developers may fully understand the possibilities of APIs. In addition to accelerating the learning curve for engineers, this method promotes a more organized and productive software development environment.

In the end, this study provides a strong basis for upcoming API documentation enhancements. A new era of precise, interactive, and user-centered documentation is being promoted in by the combination of modern web technologies and advanced computational methodologies. This all-encompassing approach will revolutionize our interactions with API documentation and propel the industry to previously unheard-of heights of efficiency, accessibility, and usability.

VII. References

[1] González-Mora, C., Barros, C., Garrigós, I., Zubcoff, J., Lloret, E., & Mazón, J. N. (2023). Improving open data web API documentation through interactivity and natural language generation. Computer Standards & Interfaces, 83, 103657.

[2] Xiao, L., & Chen, X. (2023). Enhancing LLM with Evolutionary Fine Tuning for News Summary Generation. arXiv preprint arXiv:2307.02839.

[3] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., ... & Amodei, D. (2020). Language models are few-shot learners. Advances in neural information processing systems, 33, 1877-1901.

[4] Solaiman, I., & Dennison, C. (2021). Process for adapting language models to society (palms) with values-targeted datasets. Advances in Neural Information Processing Systems, 34, 5861-5873.

[5] Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., ... & Zhou, M. (2020). Codebert: A pre-trained model for programming and natural languages. arXiv preprint arXiv:2002.08155.

[6] Moser, M., Pichler, J., Fleck, G., & Witlatschil, M. (2015, March). Rbg: A documentation generator for scientific and engineering software. In 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER) (pp. 464-468). IEEE.

[7] Shi, C., Su, Y., Yang, C., Yang, Y., & Cai, D. (2023). Specialist or Generalist? Instruction Tuning for Specific NLP Tasks. arXiv preprint arXiv:2310.15326.

[8] Zhang, C., Xiong, C., & Wang, L. (2019, August). A research on generative adversarial networks applied to text generation. In 2019 14th International Conference on Computer Science & Education (ICCSE) (pp. 913-917). IEEE.

[9] Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., ... & Sifre, L. (2022). Training compute-optimal large language models. arXiv preprint arXiv:2203.15556.

[10] Sun, C., Qiu, X., Xu, Y., & Huang, X. (2019). How to fine-tune bert for text classification?. In Chinese Computational Linguistics: 18th China National Conference, CCL 2019, Kunming, China, October 18–20, 2019, Proceedings 18 (pp. 194-206). Springer International Publishing.

[11] Lester, B., Al-Rfou, R., & Constant, N. (2021). The power of scale for parameter-efficient prompt tuning. arXiv preprint arXiv:2104.08691.

[12] Liang, P., Bommasani, R., Lee, T., Tsipras, D., Soylu, D., Yasunaga, M., ... & Koreeda, Y. (2022). Holistic evaluation of language models. arXiv preprint arXiv:2211.09110.

[13] Liao, B., Meng, Y., & Monz, C. (2023). Parameter-Efficient Fine-Tuning without Introducing New Latency. arXiv preprint arXiv:2305.16742.

[14] Feuerriegel, S., Hartmann, J., Janiesch, C., & Zschech, P. (2023). Generative AI.