what’s-up-doc: A Directory-Level Command-Line Tool for Automated Code Documentation Generation Powered by a Mixture of Open-Source LLMs

# 1. Abstract

The ongoing project represents a significant advancement in code comprehension and documentation generation via a command-line tool that is seamlessly integrated with LangChain and Large Language Models (LLMs). This tool has been painstakingly designed to address the perennial problem that developers and software engineers face: the need for comprehensive, language-agnostic, and interconnected code documentation. This project is built around the concept of LangChain, a dynamic framework that uses LLM capabilities to unlock new dimensions of code comprehension.

LangChain, in contrast to traditional tools that operate within the confines of a single programming language, transcends language barriers, making it an indispensable asset in multi-language development environments. The LangChain-powered LLM can perceive not only syntax but also semantics, relationships, and interactions in a user's codebase. Developers can start a transformative process of code documentation production by simply giving a directory path as an argument or flag. The tool should traverse the directory's files and subdirectories, deciphering the language of each code snippet and unravelling the intricate web of relationships between files and functions. It creates a detailed markdown file for each file and logical structure it encounters, exhaustively documenting the code's purpose, structure, and functionality.

This markdown page is becoming more than simply documentation; It becomes an index for every logical structure in the code base. Within this markdown file, users will find code blocks annotated with language-agnostic comments that elucidate the intent and logic behind each segment of code. It goes beyond code comments and delves into the intricacies of classes, functions, variables, and the interactions between different code files.

The result is a documentation artifact that caters to both novice developers seeking an entry point into the project and seasoned engineers striving to understand the code's intricacies. The command-line tool's versatility shines through its ability to generate documentation for projects spanning diverse programming languages. The LangChain-powered LLM adapts to the nuances of each language, providing consistent and accurate documentation that adheres to language-specific conventions and practices.

# 2. Introduction

In the ever-evolving landscape of software development, the challenge of understanding and documenting complex codebases transcends programming languages and spans the spectrum of experience levels, from novice developers to seasoned engineers. This project is born from the recognition that a groundbreaking solution is needed to bridge the gap between code and clarity, and it emerges as a pioneering endeavour at the intersection of Natural Language Processing (NLP) and command-line tools.

At its core, this project envisions a transformative command-line tool empowered by LangChain—a dynamic framework that seamlessly integrates Large Language Models (LLMs) into the code documentation process. The objective is clear: to empower developers with a tool that not only understands the intricacies of code across diverse programming languages but also comprehends the relationships between files and functions within a codebase. With a user-friendly interface, this tool opens doors to codebases of any size and complexity.

By simply specifying a directory path as a command-line argument, developers gain access to a sophisticated code understanding mechanism. The tool embarks on a journey through the directory's files and sub-directories, deciphering the languages, recognizing structures, and forging connections. The result is a meticulously crafted Markdown document—a code narrative that explains, annotates, and demystifies the codebase. It goes beyond mere code comments; it elucidates classes, functions, and variable interactions, all while accommodating the nuances of each programming language.

# 3. Literature Review

Automated code documentation generation has garnered significant attention in recent years due to its potential to enhance software development practices. In this literature review, we delve into pertinent works and advancements in this field, emphasizing innovative approaches and state-of-the-art models.

**HAConvGNN: Hierarchical Attention Based Convolutional Graph Neural Network for Code Documentation Generation in Jupyter Notebooks [1]**  
Liu et al. (2021) introduced HAConvGNN, a model tailored for generating documentation in computational notebooks like Jupyter. By incorporating a hierarchical attention mechanism, HAConvGNN addresses challenges posed by interconnected code cells, thereby enhancing the clarity and context-awareness of generated documentation.

**CodeBERT: A Pre-Trained Model for Programming and Natural Languages [2]**  
Feng et al. (2020) proposed CodeBERT, a bimodal pre-trained model bridging programming language (PL) and natural language (NL). CodeBERT's success in code documentation generation tasks showcases the potential of large language models (LLMs) in improving code comprehension and documentation practices.

**CodeTrans: Towards Cracking the Language of Silicon’s Code Through Self-Supervised Deep Learning and High-Performance Computing [3]**  
Elnaggar et al. (2021) introduced CodeTrans, an encoder-decoder transformer model tailored for software engineering tasks. Its superior performance across various software engineering tasks highlights the effectiveness of transformer models in code comprehension and documentation.

**RepoAgent: An LLM-Powered Open-Source Framework for Repository-level Code Documentation Generation [4]**  
Luo et al. (2024) presented RepoAgent, an open-source framework powered by large language models for code documentation generation at the repository level. RepoAgent showcases advancements in utilizing generative models for proactive documentation maintenance and updates.

**The Unreasonable Ineffectiveness of the Deeper Layers [5]**  
Gromov et al. (2024) empirically studied a layer-pruning strategy for popular families of open-weight pretrained LLMs, finding minimal degradation of performance until after a large fraction of the layers are removed. They identified optimal layer blocks to prune based on layer similarity and used parameter-efficient finetuning (PEFT) methods like quantization and Low-Rank Adapters (QLoRA) for "healing" the damage caused by pruning. This paper's techniques are valuable for optimizing LLMs in resource-constrained environments.

3.1. Discussion and Contribution

We acknowledge the evolution of automated code documentation generation showcased in the aforementioned works. Building upon these advancements, our project aims to contribute to this domain, leveraging techniques such as layer pruning, quantization, and Low-Rank Adapters to optimize the performance of LLMs in code documentation tasks while reducing computational resources.

3.2. Conclusion

The reviewed papers emphasize the importance of automated code documentation generation in enhancing code comprehension, maintainability, and collaboration. Our project distinguishes itself through its focus on custom model architecture and training, open-source multi-language support, advanced context understanding, documentation updates, benchmarking of LLMs, and optimization techniques based on recent studies such as "The Unreasonable Ineffectiveness of the Deeper Layers."

By ensuring high standards of accuracy, fairness, and performance in the generated documentation, our project aims to meet the evolving needs of software developers across diverse programming environments.

# 4. Problem Statement

Developers face the daunting task of documenting their code, which is often time-consuming and error-prone. Existing solutions are typically language-specific and lack flexibility. Our project addresses this challenge by developing a language-agnostic tool that automatically generates initial project documentation, offering developers a solid foundation from which they can streamline or expand their documentation efforts. By providing a universal solution compatible with mainstream programming languages, **we aim to simplify and democratize the documentation process for developers, ultimately saving time and improving code understanding**.

# 5. References

Liu, X., Wang, D., Wang, A., Hou, Y., & Wu, L. (2021, March 31). *HAConvGNN: Hierarchical Attention Based Convolutional Graph Neural Network for Code Documentation Generation in Jupyter Notebooks*. arXiv.org. <https://arxiv.org/abs/2104.01002v2>

Feng, Z., Guo, D., Tang, D., Duan, N., Feng, X., Gong, M., Shou, L., Qin, B., Liu, T., Jiang, D., & Zhou, M. (2020, February 19). *CodeBERT: A Pre-Trained Model for Programming and Natural Languages*. arXiv.org. <https://arxiv.org/abs/2002.08155v4>

Elnaggar, A., Ding, W., Jones, L., Gibbs, T., Feher, T., Angerer, C., Severini, S., Matthes, F., & Rost, B. (2021, April 6). *CodeTrans: Towards Cracking the Language of Silicon’s Code Through Self-Supervised Deep Learning and High Performance Computing*. arXiv.org. <https://arxiv.org/abs/2104.02443v2>

Luo, Q., Ye, Y., Liang, S., Zhang, Z., Qin, Y., Lu, Y., ... & Sun, M. (2024). RepoAgent: An LLM-Powered Open-Source Framework for Repository-level Code Documentation Generation. *arXiv preprint arXiv:2402.16667*.

Gromov, A., Tirumala, K., Shapourian, H., Glorioso, P., & Roberts, D. A. (2024). The Unreasonable Ineffectiveness of the Deeper Layers. *arXiv preprint arXiv:2403.17887*.