Maksym Nedoshev
PhD-student
National University of Life and Environmental Sciences of Ukraine
0009-0000-9820-0649
nedoshev@pm.me
Viktor Kyrychenko
PhD in Physical and Mathematical Sciences, Associate Professor
National University of Life and Environmental Sciences of Ukraine
0009-0001-0575-8684
v.kyrychenko@nubip.edu.ua

SEMANTIC SEARCH IN DOCUMENTATION TO ENHANCE THE EFFICIENCY OF UI COMPONENT GENERATION USING LANGUAGE MODELS

Abstract. Large software projects produce extensive documentation, which developers need to access efficiently. This paper presents a lightweight pipeline that converts documentation into a searchable knowledge base for generative LLMs in automated UI code generation. The system combines document crawling, chunking, vector embeddings, and semantic search. Evaluation on representative test cases shows that Retrieval-Augmented Generation (RAG) improves code accuracy and framework compliance compared to baseline LLM generation.

Keywords: RAG; semantic search; embeddings; documentation; LLM; UI library; code generation.

1. INTRODUCTION

Modern software development projects accumulate large volumes of documentation (guides, API references, examples). Developers need fast and precise access to relevant information.

Problem Statement. Typical large language models (LLMs) often generate inaccurate or fabricated responses when questions relate to internal or domain-specific materials [1]. This reduces their usefulness for developers. Therefore, it is reasonable to combine LLMs with a local knowledge base — the Retrieval-Augmented Generation (RAG) approach [2].

Purpose of the Publication. The purpose of this study is to present the architecture and implementation methodology of a lightweight question-answering system for project documentation based on embeddings, and to demonstrate its application in automated UI component code generation using generative LLM models.

2. RESEARCH METHODS

To evaluate the results, twelve typical code generation queries were created, and the generated code was analyzed. An example of a test description is as follows:

```
    {
      query: 'Make auth form with email and password inputs, remember me checkbox,
and submit button.',
      patterns: [
          'VaForm -> VaInput{"placeholder":"Email"}',
          'VaForm -> VaCheckbox{"label":"Remember me"}',
          'VaForm -> VaButton{"type":"submit"}',
        ]
     }
```

To validate the generated code, the proportion of abstract syntax tree (AST) patterns matched by the output was quantified. Each task comprised a code generation query alongside a set of reference patterns delineating the expected structure of the component tree. The generated code was scored based on its conformity to these predefined patterns.

3. THE RESULTS AND DISCUSSION

As a result of the study, a system was developed for integrating web framework documentation into the context of a large language model (LLM) for code generation.

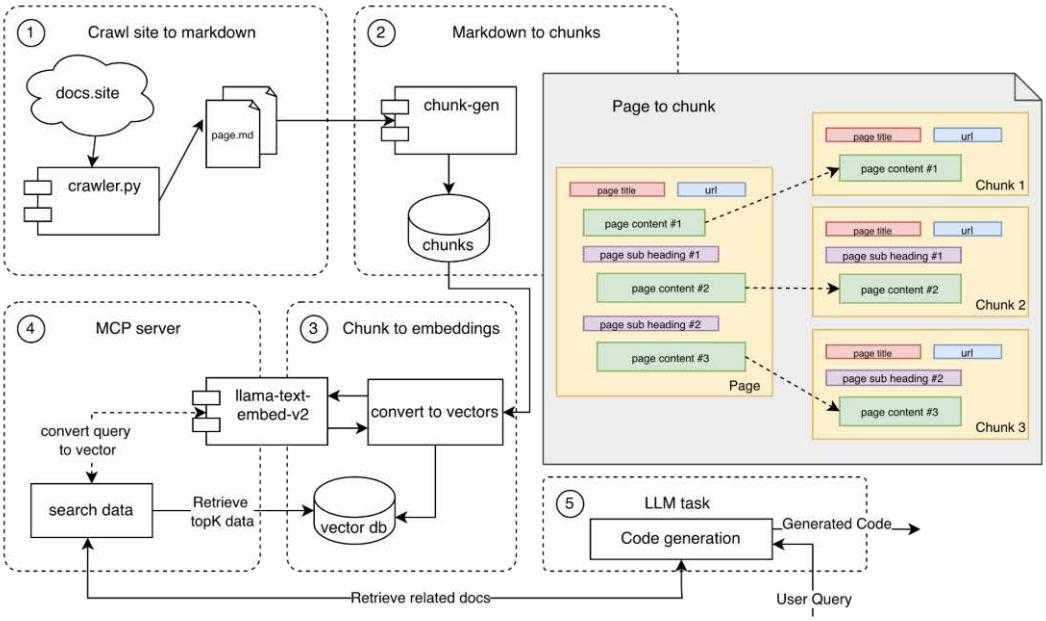

Fig. 1. Overview of Context Flow for Code Generation

The system comprises the following key components:

1. Documentation crawling and preprocessing. The documentation website was systematically crawled, and a simplified version of the content was generated in Markdown format. The implementation utilized Python libraries including BeautifulSoup, markdownify, and Playwright.
2. Segmentation into logical blocks (chunks). Documentation was partitioned into coherent logical units while preserving context, similar to human reading comprehension.
3. Vectorization of logical blocks. Each chunk was transformed into a vector representation using the llamatext-embed-v2 embedding model and subsequently stored in a vector database. In this implementation we used Pinecone service, however, any vector database, such as pgvector, is suitable.
4. MCP server implementation. The MCP server [3] was developed using the ModelContextProtocol SDK in combination with Bun.
5. Query processing and code generation. Users submit queries to the LLM via an IDE, and an agent interacts with the MCP server to generate code in accordance with the documentation context.

The system was evaluated on nine representative test cases, with the results summarized in Table 1. For each test, a manual analysis of the generated code was performed, along with an assessment of its conformity to predefined structural patterns. Without RAG integration, the generated code was simplistic and did not leverage the framework. In contrast, the RAG-enabled system allowed generation of framework-compliant code without requiring explicit specification of technologies or source references for existing components.

Table 1

Results of System Testing

Query	Result using RAG	Result without RAG
Auth form	VaForm, VaInput, VaButton, VaCheckbox, validation.	React + Material UI;
Dashboard layout	VaLayout, VaSidebar, VaNavbar, mobile-friendly.	HTML + CSS
User profile page	VaAvatar, VaInput, VaForm.	“I don’t know”.
Responsive navbar	VaNavbar, VaSidebar, v-if with mobile utilities .	HTML + CSS Media Queries.
Dark mode toggle	useColors(), VaSwitch, dynamic theme switch.	HTML + JS.
Filterable data table	VaDataTable with :items, search, sort.	HTML
Login page	VaForm, VaInput, VaButton, rules, error states.	HTML form.
Registration form	VaForm, VaInput, VaButton.	React-components using MUI.
Contact form	VaTextarea, VaButton, VaForm.	HTML

CONCLUSIONS AND PROSPECTS FOR FURTHER RESEARCH

The developed RAG system architecture for agents encompasses a fully reproducible workflow, spanning from documentation collection, structuring, and vectorization to query construction in the vector space and subsequent utilization of LLM outputs. Testing demonstrated that the integration of RAG significantly improves code generation accuracy, achieving up to 100% conformity with framework component structures, whereas baseline code generation without contextual augmentation failed to produce relevant structures.

REFERENCES

1. L. Huang, W. Yu, W. Ma, W. Zhong, “A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions,” ACM Transactions on Information Systems, vol. 43, no. 2, pp. 1–55, Jan. 2025, doi: 10.1145/3703155.
2. P. Lewis et al., “Retrieval-augmented generation for knowledge-intensive NLP tasks,” Advances in Neural Information Processing Systems, vol. 33, pp. 9459–9474, 2020.
3. “What is the Model Context Protocol (MCP)?,” Model Context Protocol. [Online]. Available: https://modelcontextprotocol.io/docs/getting-started/intro. [Accessed: 8 Nov. 2025].

MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE

NATIONAL UNIVERSITY OF LIFE AND ENVIRONMENTAL SCIENCES OF UKRAINE

FACULTY OF INFORMATION TECHNOLOGY

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

PROCEEDINGS

МАТЕРІАЛИ

XIII International scientific and practical conference

XIII Міжнародної науково-практичної конференції

GLOBAL AND REGIONAL PROBLEMS OF INFORMATIZATION IN SOCIETY AND NATURE USING '2025

ГЛОБАЛЬНІ ТА РЕГІОНАЛЬНІ ПРОБЛЕМИ ІНФОРМАТИЗАЦІЇ В СУСПІЛЬСТВІ І ПРИРОДОКОРИСТУВАННІ '2025

13-14 November 2025

Kyiv, NULES of Ukraine

13-14 листопада 2025 року

Київ, НУБіП України

Kyiv 2025

Київ 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ

ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

МАТЕРІАЛИ

XIII Міжнародної науково-практичної конференції

ГЛОБАЛЬНІ ТА РЕГІОНАЛЬНІ ПРОБЛЕМИ ІНФОРМАТИЗАЦІЇ В СУСПІЛЬСТВІ І ПРИРОДОКОРИСТУВАННІ '2025

13-14 листопада 2025 року

Київ, НУБіП України

Київ 2025

УДК 004

Рекомендовано до друку вченою радою факультету інформаційних технологій Національного університету біоресурсів і природокористування України (протокол № 4 від 18.12.2025).

Укладач: д.т.н., доцент Шкарупило В.В.

Збірник матеріалів XIII Міжнародної науково-практичної конференції "Глобальні та регіональні проблеми інформатизації в суспільстві і природокористуванні '2025", 13–14 листопада 2025 року, НУБіП України, Київ. – К.: НУБіП України, 2025. – 206 с.

Відповідальність за зміст публікацій несуть автори.