Comparative Study of Documentation Systems for CMS with an AI-Assisted Docs-as-Code Prototype

Buruj Mamdouh Kiahamy
buruj1425@icloud.com

Under the supervision of
Mehnaz Hafeez
mehnaz.hafeez@cern.ch

Abstract

Large scientific collaborations depend on extensive technical documentation. The CMS experiment at CERN has relied on TWiki for over two decades to manage approximately 44,677 documentation pages. This study investigates the modernisation of the CMS documentation ecosystem through two contributions. First, we develop a custom TWiki-to-GFM converter, twiki2gfm, implementing 96 conversion rules as a 15-phase regex pipeline. The converter achieved a 98.6% success rate on the full corpus in 24 minutes, compared with 55.9% for Pandoc. Second, we design and deploy an AI-assisted Retrieval-Augmented Generation (RAG) pipeline using the WhereIsAI/UAE-Large-V1 embedding model, ChromaDB, hybrid retrieval, cross-encoder reranking, and Qwen2.5-14B-Instruct-AWQ. The resulting prototype provides a CERN-network chat interface that generates documentation-grounded responses with source citations.

Date :28 June — 21 August 2026
Code: https://github.com/Buruj-MK/cms-twiki-assistant
/**Project Page:** https://buruj-mk.github.io/cms-twiki-assistant

Table of Contents

1 Introduction 3

2 Literature Review 3

2.1 Wiki-Based Documentation Systems **3**

2.2 Documentation Markup Languages **3**

2.3 Documentation Frameworks **3**

2.4 Semantic Search **3**

3 TWiki-to-GFM Conversion 3

3.1 Pandoc Evaluation **3**

3.2 Custom Converter: twiki2gfm **4**

3.3 Full-Corpus Batch Conversion **4**

4 AI-Based Semantic Search Pipeline 5

4.1 Pipeline Architecture **5**

4.2 Text Chunking **5**

4.3 Embedding and Vector Indexing **5**

4.4 Hybrid Retrieval and LLM **5**

4.5 User Interface and Interaction **5**

5 CERN Infrastructure and Project Resources 6

5.1 CERN Compute Infrastructure **6**

5.2 CERN OpenStack **6**

5.3 Project Storage and Data Resources **6**

5.4 Project Repositories and Links **6**

6 Results 6

7 Discussion 7

8 Limitations and Future Work 7

9 Conclusion 7

References 8

1 Introduction

The CMS experiment at CERN is one of the largest scientific collaborations in the world, involving thousands of physicists, engineers, and computing experts from hundreds of institutions worldwide [1]. Since the early 2000s, CMS has relied on the TWiki platform [2] to document detector operations, software procedures, computing workflows, and collaboration policies. Over more than two decades of continuous use, this repository has grown to approximately 44,677 interconnected documentation pages.

While TWiki enabled effective collaborative editing during the early stages of the experiment, the accumulated documentation now presents significant challenges: fragmented organisation across hundreds of webs, inconsistent formatting due to evolving markup conventions, keyword-only search that fails when collaborators use different terminology for the same concepts, and no integration with the Git-based workflows already standard for CMS software development.

The increasing adoption of the *Documentation as Code* paradigm [3] has motivated the investigation of modern Git-based documentation ecosystems. This project addresses four objectives: (1) comparative evaluation of documentation markup languages and frameworks, (2) empirical assessment of Pandoc as a migration tool, (3) development and production-scale deployment of a custom converter, and (4) design and deployment of an AI-enabled semantic search pipeline using embedding models and Retrieval-Augmented Generation (RAG).

2 Literature Review

2.1 Wiki-Based Documentation Systems

Three major wiki platforms were studied. **TWiki** [2] is an open-source enterprise wiki developed in 1998 using Perl, providing collaborative editing and fine-grained access control. CMS adopted it during detector construction. **MediaWiki** [4], the engine behind Wikipedia, demonstrates scalability but shares TWiki's limitations in version control integration. **Confluence** [5] by Atlassian offers a modern interface but requires commercial licensing unsuitable for open scientific collaborations. All three platforms lack native integration with Git-based development workflows and CI/CD pipelines.

2.2 Documentation Markup Languages

Six markup languages were compared: Standard Markdown [6], CommonMark [7], GitHub Flavored Markdown (GFM) [8], MDX [9], reStructuredText [10], and AsciiDoc [11]. **GFM** was selected for four reasons: (1) native rendering on GitLab, (2) support by both MkDocs and Docusaurus, (3) robust Pandoc support, and (4) minimal learning curve for CMS collaborators. A critical finding was that **MDX is fundamentally incompatible with MkDocs** because MDX requires the JavaScript/React ecosystem while MkDocs is Python-based.

2.3 Documentation Frameworks

Three frameworks were evaluated. **MkDocs** [12] was selected over Docusaurus [13] and Sphinx [10] for its lightweight Python architecture, minimal configuration, and natural integration with CERN's GitLab infrastructure. Docusaurus offers richer interactivity but introduces React/JavaScript complexity unnecessary for scientific collaboration. Sphinx targets Python-centric documentation.

2.4 Semantic Search

Traditional keyword-based search relies on exact word matching, failing when users and document authors use different terminology. **Semantic search** converts queries and documents into dense vector representations (embeddings) that capture meaning. The **Universal AnglE Embedding (UAE-Large-V1)** model [14] generates these embeddings for semantic similarity. **ChromaDB** [15] provides vector search with metadata filtering. Hybrid retrieval combines dense vectors with BM25 sparse matching via Reciprocal Rank Fusion (RRF). Cross-encoder rerankers [16] refine results jointly. **RAG** [17] passes retrieved chunks to an LLM for grounded answer generation. CERN's **AccGPT** [18] and **Archi** [19] frameworks served as reference implementations.

3 TWiki-to-GFM Conversion

3.1 Pandoc Evaluation

Pandoc [20] was initially identified as the primary migration tool due to its built-in TWiki reader and GFM writer. Testing against real CMS TWiki files using the command;

pandoc input.txt -f twiki -t gfm -o output.md, revealed critical failures:

1. **Metadata lines not removed**
   TWiki files contain system metadata (%META: TOPICINFO%, %META:TOPICPARENT%) that must be stripped during conversion. Pandoc preserves this as visible raw text.
2. **WikiWords converted to broken links.**
   TWiki auto-links CamelCase words. In CMS documentation, terms like AK4PF, AK4PFHLT, and AK8CaloHLT are physics detector identifiers, not page references. Pandoc converts every WikiWord into a Markdown hyperlink (e.g., [AK4PF](AK4PF)), producing broken links to non-existent files.
3. **Table formatting is broken.**
   CMS documentation uses data-heavy tables with TWiki rowspan (|^|) and colspan (||). Pandoc passes the caret character through as literal text.
4. **TWiki variables unresolved.**

The %TOC% variable is converted to an empty HTML span; other variables (%ATTACHURL%, %INCLUDE{...}%) pass through unconverted.

5. **Bold/italic semantic inversion.**
   TWiki *text* produces bold, but in Markdown *text* produces italic — the single most critical difference. CERN runs an older TWiki version (TextFormattingRules revision 54, from 2011), introducing additional edge cases.

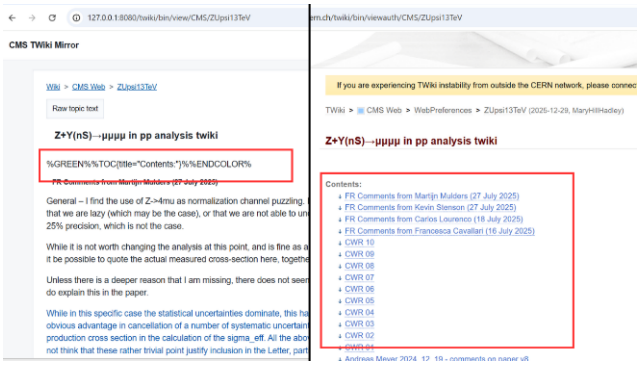


Figure 1. Example of inconsistent table-of-contents rendering between an original TWiki page and its Pandoc-converted output, illustrating the lack of consistent formatting rules across legacy TWiki documentation.

3.2 Custom Converter: twiki2gfm

A comprehensive TWiki-to-GFM conversion table was developed containing **96 individual conversion rules** across **15 categories** by studying the official TWiki TextFormattingRules documentation alongside the actual CERN TWiki instance. Based on this specification, a pure-Python CLI converter (*twiki2gfm.py*) was developed with zero external dependencies, ensuring deployment across all CERN environments including LXPLUS.

The converter implements a **15-phase regex pipeline** where each phase addresses a specific category:

1. Extract verbatim/pre blocks
2. Strip META lines
3. Color blocks
4. TWiki variables
5. Headings
6. Tables
7. Lists
8. Links
9. Inline formatting
10. WikiWord escaping
11. HTML tags
12. Signatures
13. Restore verbatim blocks
14. Horizontal rules
15. Final cleanup

The phase ordering prevents rule conflicts verbatim content is protected before any formatting rules execute. The converter is validated by **91 unit tests** using pytest across all categories. A companion tool (*report.py*) generates detailed per-file conversion reports.

3.3 Full-Corpus Batch Conversion

The full CMS TWiki corpus of **44,677 topic files** was processed through both Pandoc (v3.6.4, with a 30-second timeout per file) and twiki2gfm. Table I presents the comparison.

Table 1. Full-corpus conversion performance.

Metric	Pandoc	twiki2gfm
Files Converted	24,969 (55.9%)	44,060 (98.6%)
Errors/Timeouts	~19,708 (44.1%)	501 (1.1%)
Warnings	N/A	116 (0.3%)
Processing Time	>4 days	24 minutes
Files/Second	~0.07	~31

Pandoc's 44.1% failure rate is caused by its parser entering non-terminating states on Review-type pages containing deeply nested tables. The twiki2gfm regex architecture guarantees linear-time processing regardless of markup complexity, eliminating the timeout problem entirely.

Examples of TWiki conversion differences between Pandoc and twiki2gfm: Figure 2 shows malformed table output and conversion artefacts produced by Pandoc, while Figure 3 shows TWiki-specific metadata retained by Pandoc but removed by the custom converter.

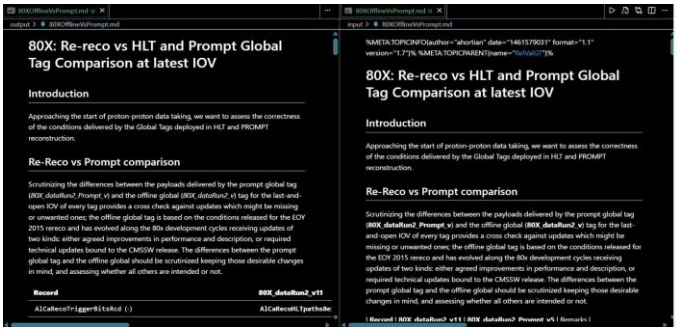


Figure 2. Example of TWiki conversion differences between Pandoc and twiki2gfm; malformed table output and conversion artefacts in Pandoc

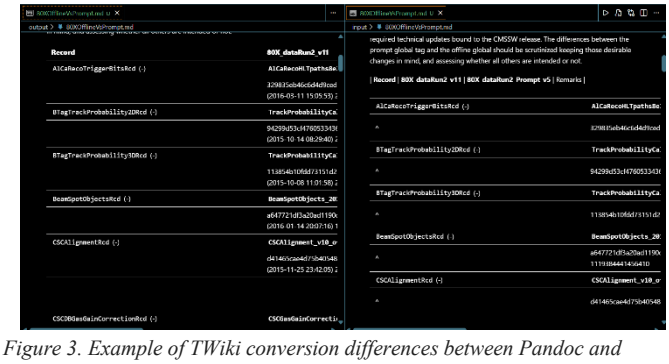


Figure 3. Example of TWiki conversion differences between Pandoc and twiki2gfm; TWiki-specific metadata retained by Pandoc but removed by the custom converter

4 AI-Based Semantic Search Pipeline

4.1 Pipeline Architecture

The proposed architecture of the CMS documentation system is presented in Figure 4. The workflow consists of two connected parts: documentation conversion and AI-assisted semantic search. In the first part, the CMS TWiki documentation is converted into GitHub-Flavored Markdown (GFM) using the custom twiki2gfm converter and stored in a Git repository. The converted documentation then feeds the second part, where it is chunked, embedded, indexed in ChromaDB, and retrieved using hybrid search and reranking before being provided to the LLM for generating source-cited responses through the chat interface.

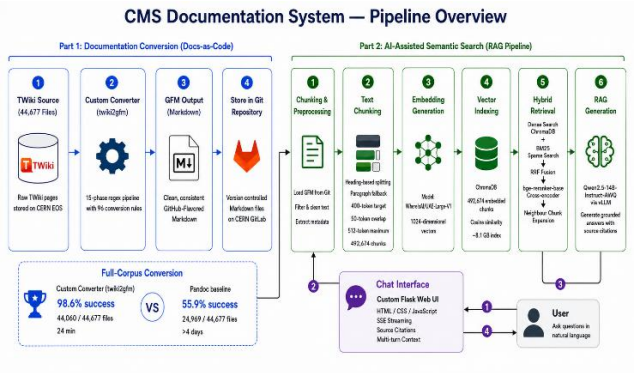


Figure 4. Overall CMS documentation modernisation pipeline, showing the TWiki-to-GFM conversion workflow and the subsequent AI-assisted semantic search and RAG pipeline.

4.2 Text Chunking

The 44,060 GFM files produced by twiki2gfm were processed by *chunker.py* implementing a three-phase pipeline: Filter → Clean → Chunk. The chunker splits primarily by heading boundaries with fallback to paragraph boundaries. Configuration: 400-token target, 50-token overlap, 512-token hard maximum. Result: **492,674 chunks** (median ~344 tokens, none exceeding 512) stored as a 905 MB JSONL file. Each chunk carries metadata including source file, page title, section heading, chunk index, author, date, and content type.

4.3 Embedding and Vector Indexing

The final embedding model is **WhereIsAI/UAE-Large-V1** [14], producing 1024-dimensional vectors. The model was selected after evaluating alternatives including all-MiniLM-L6-v2 (384 dimensions, for prototyping) and e5-large-v2 (1024 dimensions, used by CERN's AccGPT). All 492,674 chunks were embedded in batches of 256 on GPU and indexed in **ChromaDB** [15] running on vocmsgpu.cern.ch (port 8000). The complete vector index is approximately 8.1 GB. A pysqlite3-binary fix was required for ChromaDB's SQLite3 version dependency on the CERN VM. All data and virtual environments reside under /var/tmp/ due to AFS home directory quota limits.

4.4 Hybrid Retrieval and LLM

The RAG service (port 8080, served via uvicorn) implements hybrid retrieval:

1. Embed the user query with UAE-Large-V1
2. Dense vector search against ChromaDB
3. BM25 sparse keyword matching in parallel
4. Merge via **Reciprocal Rank Fusion** (RRF)
5. Rerank with **bge-reranker-base** [16] cross-encoder
6. Expand context with neighbouring chunks from the same document.

The LLM is **Qwen2.5-14B-Instruct-AWQ** [21] served via **vLLM** [22] on port 8001 with half-precision inference on Tesla T4 GPUs. Answers are streamed via Server-Sent Events using a custom format ({"type": "sources"}, {"type": "token"}, {"type": "done").

4.5 User Interface and Interaction

Open WebUI was initially evaluated but encountered a compatibility issue: it sends *tool_choice* parameters that vLLM rejects. A custom Flask-based chat interface was developed with a **sources panel** displaying retrieved TWiki pages with relevance scores, **multi-turn context** (last 3 Q&A pairs prepended as plain text), The interface includes a **feedback** mechanism that records whether users find generated responses helpful.
(thumbs-up/down saved to feedback.jsonl), and **Stats/About panels**.

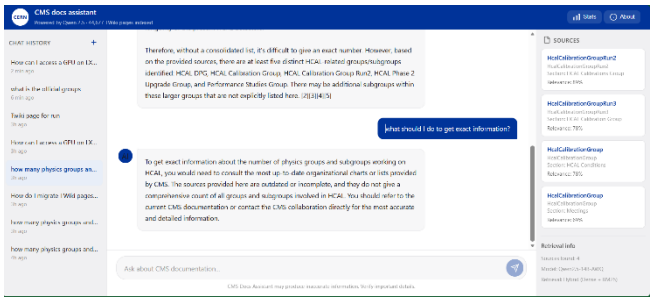


Figure 5. CMS Docs Assistant

Key interaction features of the CMS Docs Assistant shown in Figure 5 are: (a) main interface with chat history and suggested queries; (b) multi-turn interaction using previous conversational context; and (c) structured technical responses with formatted commands and code blocks.

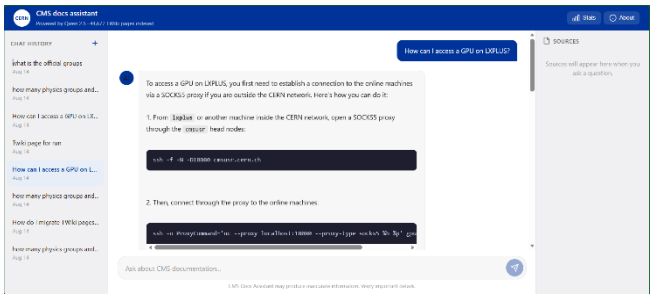


Figure 6. CMS Docs Assistant during a multi-turn interaction.

The interface maintains conversational context from previous exchanges while displaying the retrieved source documents and their relevance scores alongside the generated response, refer Figure 6.

5 CERN Infrastructure and Project Resources

5.1 CERN Compute Infrastructure

The system is deployed across two CERN virtual machines. The GPU backend, vocmsgpu.cern.ch, provides two NVIDIA Tesla T4 GPUs and hosts ChromaDB, the RAG service, and vLLM. The frontend, cms-twiki-assistant.cern.ch, hosts the custom Flask web interface and communicates with the backend through an SSH tunnel.

5.2 CERN OpenStack

The frontend VM was provisioned through CERN OpenStack using the ALMA9 operating-system image and the m2.small flavor. It hosts the custom Flask interface on port 8000.

5.3 Project Storage and Data Resources

The original TWiki corpus is stored on CERN EOS, while project resources and runtime data are maintained on the GPU backend under /var/tmp/. The chunk dataset contains 492,674 chunks (~905 MB), and the ChromaDB index is approximately 8.1 GB, refer Table 2.

Table 2. Project storage and data resources

Resource	Purpose	Location / Key Details
CMS TWiki corpus	Original documentation	CERN EOS: /eos/cms/store/group/internal_comm/twiki/CMS/topics/
Project workspace	Scripts, environments, project data	vocmsgpu:/var/tmp/
Chunk dataset	Chunked documentation	vocmsgpu:/var/tmp/embedder/chunks.jsonl; 492,674 chunks; ~905 MB
Chroma DB	Vector index	vocmsgpu:/var/tmp/chroma_db/; ~8.1 GB
EOS backup	Project backup	CERN EOS

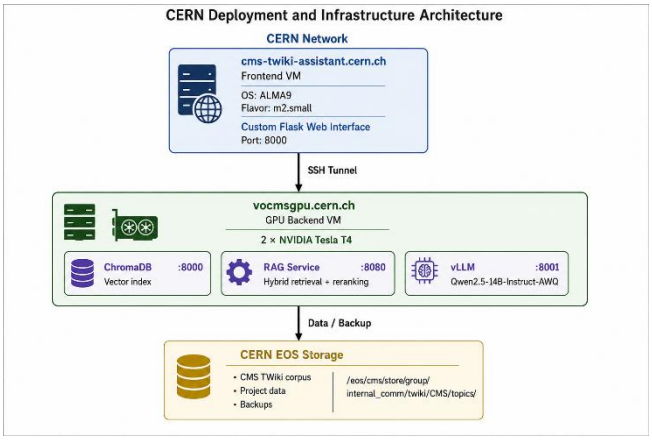


Figure 7. Deployment architecture of the CMS Docs Assistant across CERN infrastructure.

Deployment architecture of the CMS Docs Assistant across CERN infrastructure is shown in Figure 7. The frontend Flask application communicates with the GPU-backed RAG services through an SSH tunnel, while CERN EOS provides persistent storage for the documentation corpus and project data.

5.4 Project Repositories and Links

The main project repositories, infrastructure resources, and deployment endpoints are listed in Table 3. Project-specific URLs should be replaced with the verified links before submission.

Table 3. Project repositories and infrastructure resources.

Resource	Link / Path
Project GitLab repository	[Insert verified project GitLab URL]
Project page	[Insert verified project page URL]
CERN OpenStack	[Insert verified OpenStack URL]
CMS TWiki source	/eos/cms/store/group/internal_comm/twiki/CMS/topics/
EOS project backup	[Insert verified EOS backup path]
Frontend VM	cms-twiki-assistant.cern.ch
GPU VM	vocmsgpu.cern.ch

6 Results

The twiki2gfm converter achieved **98.6%** clean success on the full 44,677-file corpus in **24 minutes** (~31 files/second), compared to Pandoc's 55.9% over more than four days. The most frequently triggered conversion rules: code spans (100.0%), META stripping (99.9%), headings (78.4%), variables (73.7%), italic (68.1%), bullet lists (62.5%), signatures (60.6%), links (58.0%), bold (46.7%).

The RAG system was deployed with all **492,674 chunks** indexed in ChromaDB (~8.1 GB). The chat interface generates streaming LLM-generated answers with source citations and

relevance scores. The system correctly **scopes responses to CMS documentation** — when asked questions outside TWiki content, it responds that its knowledge is limited to CMS TWiki pages (Fig. 1 demonstrates this behaviour with the feedback mechanism).

7 Discussion

The results demonstrate that a domain-specific, regex-based converter significantly outperforms a general-purpose tool when applied to a large, heterogeneous TWiki corpus. Pandoc's TWiki reader encounters fundamental limitations with the structural complexity present in production CMS documentation, entering non-terminating states on certain table structures. The twiki2gfm regex architecture guarantees linear-time processing, eliminating this problem entirely.

The RAG pipeline demonstrates the feasibility of AI-assisted semantic search for CMS documentation and provides a retrieval mechanism that combines dense and sparse search with reranking.

The hybrid retrieval strategy addresses the vocabulary mismatch problem inherent in a large, multi-domain physics collaboration. The decision to use converted GFM files directly as the RAG data source with no separate cleaning step proved beneficial: Using the converted GFM as the RAG input avoids carrying TWiki-specific markup into the chunking and embedding stages.

8 Limitations and Future Work

This system is a prototype. There are no per-user accounts. Chat history is shared and visible to anyone who accesses the URL. The interface has no authentication, no persistent user storage, and no administrative controls. Any person on the CERN network who opens the link sees the same shared chat history.

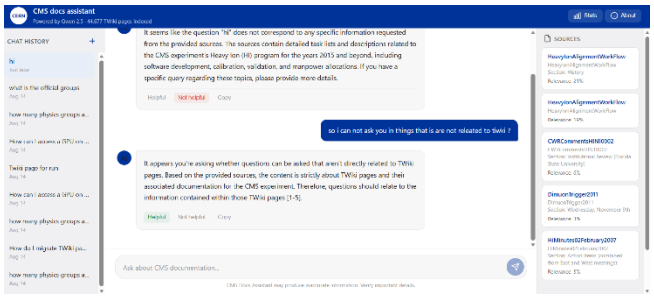


Figure 8. Example of an out-of-scope query.

The assistant is designed to answer questions grounded in the CMS TWiki documentation corpus rather than functioning as a general-purpose conversational model, refer Figure 8.

Limitation that are identified are as follows;

1. Converter coverage: The 1.1% error rate (501 files) involves non-standard <noautolink> tag nesting. The 0.3% warnings are custom TWiki variables outside the standard mapping.
2. No formal retrieval evaluation: An evaluation plan was defined (20–30 queries with ground-truth sources, measuring Hit Rate@K and Context Recall) but not executed within the internship timeframe.
3. Security: The interface runs over HTTP without authentication. HTTPS and CERN SSO integration were deferred.

Future work includes executing the formal evaluation plan; investigating alternative RAG architectures (GraphRAG, Agentic RAG); integrating with CERN's LiteLLM Gateway for access to hosted LLMs; Docker containerisation; HTTPS and CERN SSO integration; implementing per-user chat sessions; and establishing a permanent service owner within CMS Computing Operations.

9 Conclusion

This project delivers a complete system for updating the CMS documentation ecosystem at CERN. The twiki2gfm converter achieves 98.6% success on 44,677 TWiki pages in 24 minutes, providing a practical migration path. The AI-powered semantic search pipeline with no separate cleaning step transforms the converted documents into a queryable knowledge base with 492,674 indexed chunks, hybrid retrieval, cross-encoder reranking, and LLM-generated answers with source citations. The deployed prototype at cms-twiki-assistant.cern.ch demonstrates that AI-assisted semantic search is viable for large-scale scientific collaborations.

References

[1] CMS Collaboration. “The CMS Experiment at the CERN LHC.” J. Instrum. 3, S08004, 2008.
[2] TWiki — the Open-Source Enterprise Wiki. https://twiki.org/.
[3] A. Gentle. Docs Like Code. 2nd ed. Lulu.com, 2017.
[4] MediaWiki. https://www.mediawiki.org/.
[5] Atlassian. Confluence. https://www.atlassian.com/software/confluence.
[6] J. Gruber. Markdown. https://daringfireball.net/projects/markdown/.
[7] CommonMark Spec. https://commonmark.org/.
[8] GitHub Flavored Markdown Spec. https://github.github.com/gfm/.
[9] MDX Documentation. https://mdxjs.com/.
[10] Sphinx Documentation. https://www.sphinx-doc.org/.
[11] AsciiDoc Language Documentation. https://asciidoc.org/.
[12] MkDocs Documentation. https://www.mkdocs.org/.
[13] Docusaurus Documentation. https://docusaurus.io/.
[14] X. Li, J. Li. “UAE: Angle-optimized Text Embeddings.” arXiv:2309.12871, 2023.
[15] Chroma — the open-source embedding database. https://www.trychroma.com/. 2024.
[16] S. Xiao et al. “C-Pack: Packaged Resources for General Chinese Embedding.” arXiv:2309.07597, 2023.
[17] P. Lewis et al. “Retrieval-Augmented Generation for NLP Tasks.” NeurIPS 33, 2020, pp. 9459–9474.
[18] F. Rehm. AccGPT. CERN Internal, gitlab.cern.ch/dsb/accgpt-v2. 2024.
[19] Archi. https://archi-physics.github.io/archi/. 2024.
[20] J. MacFarlane. Pandoc. https://pandoc.org/.
[21] A. Yang et al. “Qwen2.5.” arXiv:2412.15115, 2024.
[22] W. Kwon et al. “vLLM: PagedAttention.” arXiv:2309.06180, 2023.
[23] N. Reimers, I. Gurevych. “Sentence-BERT.” Proc. EMNLP-IJCNLP, 2019, pp. 3982–3992.