MINUT 2026 (8), s. 31–37
ISSN 2719-3063

Marcin RYT

[1]Faculty of Applied Mathematics, Silesian University of Technology, Kaszubska 23, 44-100 Gliwice, Poland

W Machine AI assistant architecture

Abstract. This paper presents the W Machine chatbot, an intelligent tutoring layer for an educational processor platform. The chatbot ingests domain sources, embeds their content, retrieves relevant fragments via a vector database, and generates context-aware responses using a large language model. To address the shortcomings of general-purpose large language models in specialized educational settings, the proposed system employs a Retrieval-Augmented Generation architecture. The system consists of two AI-driven scenarios: a workflow linking prompt-based reasoning to automated W code generation and a fine-tuned model providing enhanced, domain-specific explanations. The RAG-based system achieved the most balanced results, offering better performance, better scalability, and easier long-term maintenance through knowledge base updates rather than repeated retraining. The findings suggest that retrieval-augmented architectures are particularly well suited to educational applications in narrow, low-level domains such as the W Machine.
Keywords: Intelligent tutoring system, Retrieval-Augmented Generation, AI-assisted programming

1. Introduction

The landscape of computer science education and software development is undergoing a fundamental shift driven by the integration of Artificial Intelligence. In recent years, AI assistants have evolved from simple autocompletion tools to sophisticated partners capable of reasoning through complex logic, debugging intricate code, and explaining abstract architectural concepts. For students and developers alike, these tools serve as a digital tutor, reducing the cognitive load associated with learning new programming paradigms and hardware architectures.

However, the effectiveness of generic Large Language Models (LLMs) is often limited when applied to highly specialized or educational domains, such as the W Machine [1], a specific instructional processor platform. Standard models frequently suffer from hallucinations or lack the precise, up-to-date technical context required to provide accurate guidance on unique instruction sets or low-level configurations. To

Corresponding author: M. Ryt (mr303254@student.polsl.pl).
Received: 02.03.2026.

bridge this gap, modern AI implementations must move beyond general knowledge and toward grounded reasoning.

This article introduces the W Machine chatbot, an intelligent tutoring layer designed to provide precise, context-aware assistance. Using a Retrieval-Augmented Generation (RAG) [2] architecture supported by efficient indexing tools such as Facebook AI Similarity Search (FAISS) [3], the system ensures that every response is anchored in official documentation and verified domain sources. Rather than relying on the model's static memory, the assistant dynamically retrieves relevant knowledge fragments at query time using advanced models such as *Gemini 2.5 Flash* [4] and *Qwen3 Embedding* [5], offering a level of accuracy and traceability essential for an educational environment.

2. Technical Constraints and Implementation Challenges of the W Machine

The W Machine is a simplified educational computer model used to introduce fundamental concepts of computer architecture. It is designed to make the internal operation of a processor explicit, allowing direct observation of instruction execution, data movement, and control flow. Owing to its minimal and deterministic structure, the W Machine serves as a clear and accessible platform for understanding the relationship between abstract architectural principles and their concrete hardware realization.

The W Machine constitutes a simplified implementation of the von Neumann architecture, designed to expose instruction-level execution mechanisms [1]. In contrast to contemporary processor architectures that incorporate multiple registers, advanced addressing modes, and hardware-supported control structures, the W Machine provides only a minimal architectural model. Consequently, software development in this environment requires direct manipulation of processor state, memory contents, and instruction sequencing, significantly increasing the complexity of implementation.

The lack of automated mechanisms for register allocation, control flow abstraction, and memory protection transfers full responsibility for correctness to the programmer. Algorithmic constructs must therefore be expressed as explicit low-level operations that closely correspond to the underlying hardware behavior. Although such an approach improves architectural transparency, it simultaneously introduces substantial challenges during program development and verification.

2.1. Data Processing Constraints

A primary source of programming difficulty in the W Machine arises from constraints imposed on data processing. The architecture relies on a single accumulator register as the exclusive location for arithmetic and logical operations. As a result, intermediate values cannot be retained within the processor and must be stored and retrieved repeatedly from the main memory.

This limitation necessitates the decomposition of even simple computations into extended sequences of elementary operations, each carefully ordered to preserve the correct accumulator state. Programs become highly sensitive to instruction ordering, as unintended modifications of the accumulator may invalidate subsequent computations. The increased frequency of explicit memory accesses leads to longer instruction sequences and reduced program clarity, while also increasing the risk of implementation errors.

2.2. Control Flow and Memory Interaction Limitations

Additional complexity stems from the absence of native support for high-level control flow constructs. Program execution is governed exclusively by explicit manipulation of the program counter and conditional branching based on processor state information, specifically the sign of the accumulator value. Therefore, high-level conditional logic must be reformulated into arithmetic evaluations that influence execution flow.

Furthermore, the lack of indirect addressing and memory protection mechanisms complicates coordination between code and data. Operations on sequential memory locations or data collections require careful manual management of memory addresses. Techniques involving the modification of instruction operands during execution increase program fragility, as minor address miscalculations may result in unintended overwriting of instruction words.

3. System Architecture

To overcome the limitations of generic large language models in specialized educational settings, the proposed W Machine chatbot is built upon a RAG [2] architecture. Rather than relying solely on the internal parameters of the model, the system integrates an external knowledge base with semantic retrieval and controlled generation. This approach allows the chatbot to provide precise, verifiable, and contextually grounded responses, which is particularly important in educational scenarios where accuracy and traceability are critical. Unlike conventional LLMs that may generate plausible but unsupported answers, the RAG framework ensures that every response is either substantiated by authoritative sources or explicitly signals missing information.

3.1. Knowledge Base Construction

The knowledge base contains curated domain-specific materials related to the W Machine [1], including documentation, instructional texts, architectural descriptions, and worked solutions. To support efficient retrieval and higher semantic precision, the corpus is not treated as a single monolithic document. Instead, it is preprocessed into smaller, semantically coherent chunks.

A hybrid chunking methodology is employed to achieve this segmentation. Initially, a recursive text splitter is applied to decompose documents along natural structural boundaries such as sections, paragraphs, and sentences. This step preserves syntactic correctness and logical flow while preventing arbitrary text fragmentation. To further refine the quality of the chunks, the *Gemini 2.5 Flash* model [4] is used to assess contextual coherence and semantic continuity across candidate segments. By using the model's language understanding capabilities, the system ensures that each chunk encapsulates a complete and contextually self-consistent idea rather than isolated or ambiguous text fragments.

Each chunk is then transformed into a high-dimensional vector representation using the *Qwen3-Embedding-0.6B* model [5]. These embeddings act as semantic fingerprints, capturing meaning beyond surface-level keywords. All embeddings are computed offline and stored persistently, enabling fast and consistent retrieval during runtime.

3.2. Retrieval Layer: Semantic Search with FAISS

The retrieval layer is responsible for selecting the most relevant knowledge chunks in response to a user query. Unlike traditional keyword-based search, the system relies on semantic similarity in the embedding space.

When a user submits a question, it is embedded using the same *Qwen3-Embedding-0.6B* model [5] to ensure representational consistency. Relevance between the query and stored chunks is evaluated using cosine similarity, which measures the angular distance between vectors and effectively captures conceptual alignment.

To support efficient nearest-neighbor search at scale, the system employs Facebook AI Similarity Search (FAISS) [3]. FAISS provides optimized indexing structures and algorithms for high-dimensional vector search, allowing the system to rapidly identify the top-k most relevant chunks even as the knowledge base grows.

If the retrieval stage yields no results, or if all candidate chunks fall below a predefined similarity threshold, the system explicitly treats the query as unsupported by the knowledge base. In such cases, no context is forwarded to the generation layer, and the system signals a controlled "lack of information" condition instead of attempting speculative generation.

3.3. Generation Layer: Context-Grounded Response Synthesis

Once the most relevant knowledge chunks are retrieved, they are passed to the generation layer, which is responsible for producing the final user-facing response. Text generation is performed using the *Gemini 2.5 Flash* model [4].

The model is explicitly instructed to base its responses on the retrieved context. Rather than assuming access to complete domain knowledge, the model operates under the constraint that the provided context represents the authoritative and exhaustive information available for the current query. When the retrieved context is insufficient or empty, the model is instructed to clearly communicate this limitation, for example, by indicating that the requested information is not present in the available materials.

A structured prompting approach reinforces this behavior.

Each prompt includes:

- Explicit instructions defining the model's role, tone, and grounding constraints.
- The retrieved knowledge chunks, treated as authoritative context.
- The user's question.
- Optionally, recent conversational history to maintain continuity.

3.4. End-to-End RAG Workflow

From an external perspective, user interaction with the chatbot is seamless. Internally, each query follows a well-defined RAG [2] pipeline (Fig. 1):

1. The system receives the user's question via the API.
2. An embedding of the question is computed using *Qwen3-Embedding-0.6B* [5].
3. FAISS retrieves the top-k most semantically relevant knowledge chunks using cosine similarity [3].

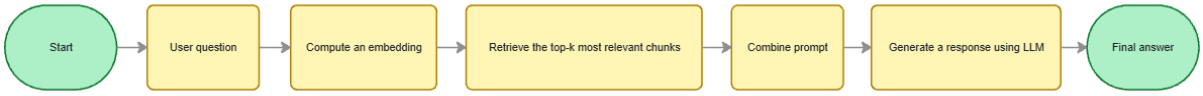

Figure 1. Structure of RAG workflow

4. A grounded prompt is assembled from system instructions, retrieved context, and the user's query.
5. The *Gemini 2.5 Flash* model generates a response constrained by the provided context [4].
6. If no sufficiently relevant context is available, the system returns an explicit notification of missing information.
7. The final answer is returned to the client.

This unified architecture ensures that each response is traceable to specific instructional sources, auditable, and robust against unsupported or out-of-scope queries, making the system suitable for high-stakes educational support.

4. Experimental Approaches to W Machine Assistance

At the outset, we organized the implementation around three parallel approaches for assisting users with W Machine programming tasks. The first approach uses a local language model fine-tuned on W Machine-specific data to provide domain-aware explanations, guidance, and code suggestions. The second approach combines an LLM with an automated transpiler, introducing Python as an intermediate representation to structure reasoning and reduce translation errors. Later, both approaches were compared with RAG system to assess their relative effectiveness and scalability.

4.1. Combining transpiler and LLM

This approach implements a multi-stage transformation pipeline that combines a Large Language Model with an automated transpiler. Rather than generating W Machine instructions directly from natural language, the process introduces Python as an intermediate representation to structure reasoning and reduce translation errors.

In this workflow, the *Qwen3 4B* [5] model translates user prompts into structured Python code that captures the intended algorithm and control flow, leveraging Python as a high-level, human-readable abstraction that LLMs handle more effectively than W Machine instructions, producing more accurate and consistent outputs [6].

The generated Python code is then passed to a deterministic transpilation stage, which converts it into W Machine instructions. This transpiler systematically maps Python constructs to their low-level equivalents, ensuring that the final output adheres strictly to the W Machine's execution model.

4.2. Fine-Tuned LLM for Domain-Specific W Machine Support

In this approach, local LLMs were fine-tuned on W Machine-specific data to provide domain-aware explanations, guidance, and code suggestions. We experimented with the *Qwen3-4B* [5] and *Gemma3-4B* [7] models. While fine-tuning improved contextual accuracy compared to generic models, results were

mostly reliable only for simple tasks due to the limited size of the training corpus and local computational constraints.

4.3. Testing Methodology and Comparative Cases

The three approaches were systematically evaluated to quantify their effectiveness in supporting users with W Machine programming tasks. Preliminary experiments revealed that the locally fine-tuned LLM offered sufficient guidance only for relatively simple exercises, a limitation primarily attributable to the restricted size of the training dataset and the constrained computational resources available for inference. In contrast, the transpilation pipeline augmented with LLM support yielded more consistent outcomes for well-structured tasks, whereas the RAG-based system demonstrated robust performance across a broader spectrum of use cases.

Among the evaluated methods, the RAG approach achieved the highest overall performance. This is largely due to its use of the advanced *Gemini* language model [4], which integrates the retrieval of authoritative W Machine documentation with high-quality, context-aware text generation. By explicitly combining knowledge retrieval with generative modeling, the RAG system not only produces more accurate and reliable mappings of user intents to W Machine instructions, but also establishes a more scalable and maintainable framework than the fine-tuned or transpiler-based alternatives. In particular, new domain knowledge can be incorporated by updating the underlying knowledge base, obviating the need for retraining the core language model.

5. Summary

This paper describes the design and evaluation of an AI-assisted tutoring system intended to support both learning and software development for the W Machine [1]. Due to its minimalist von Neumann architecture, single-accumulator processing model, and absence of high-level control flow mechanisms, programming the W Machine poses considerable challenges for students. Effective use of the platform requires careful reasoning about instruction order, memory usage, and processor state, which often proves difficult for beginners.

To address the shortcomings of general-purpose large language models in this highly specialized educational setting, the proposed system employs a RAG [2] approach. A curated, domain-specific knowledge base is combined with semantic retrieval implemented using FAISS [3] and dense embeddings produced by the *Qwen3-Embedding* model [5]. This design ensures that generated responses are firmly grounded in official documentation and technical references, reducing the risk of unsupported or misleading explanations. The response generation itself is handled by the *Gemini* language model [4], which is constrained to operate strictly on the retrieved material.

The study also examined three different strategies for providing automated assistance for the W Machine: fine-tuned local language models, an LLM-supported transpilation pipeline using Python as an intermediate representation, and the final RAG-based solution. Experimental results showed that fine-tuned models perform adequately only for relatively simple tasks and are limited by training data availability and computational cost. The transpilation-based approach improves reliability for well-structured problems but lacks flexibility. The RAG-based system, by contrast, achieved the most balanced results, offering higher accuracy, better scalability, and easier long-term maintenance through knowledge base updates rather than repeated retraining.

Overall, the findings suggest that retrieval-augmented architectures are particularly well suited to educational applications in narrow, low-level domains such as the W Machine. By combining verified knowledge retrieval with carefully constrained language generation, the proposed tutoring system provides accurate, transparent, and pedagogically meaningful support, forming a strong basis for future intelligent tools in computer architecture education.

Acknowledgment

The author would like to thank Robert Tutajewicz and Krzysztof Simiński for their guidance, valuable discussions, and support during the development of the presented system.

References

1. A. Momot, R. Tutajewicz, *Maszyna W jak zaprojektować prosty rozkaz*, MINUT 2019 (1), pp. 24-35.
2. P. Lewis, E. Perez, A. Piktus, et al., *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*, Advances in Neural Information Processing Systems (NeurIPS), vol. 33, pp. 9459–9474, 2020.
3. M. Douze, A. Guzhva, C. Deng, et al., *The Faiss library*, IEEE Transactions on Big Data, vol. 12, no. 2, pp. 346-361, April 2026
4. Gemini Team, Google, *Gemini 2.5: Pushing the Frontier with Advanced Reasoning, Multimodality, Long Context, and Next Generation Agentic Capabilities*, 2025 (arXiv:2507.06261)
5. Qwen Team, Alibaba Group, *Qwen3 Embedding: Advancing Text Embedding and Reranking Through Foundation Models*, 2025 (arXiv:2506.05176)
6. L. Twist, J. M. Zhang, M. Harman, et al., *A Study of LLMs' Preferences for Libraries and Programming Languages*, 2025 (arXiv:2503.17181)
7. Gemma Team, Google DeepMind, *Gemma 3 Technical Report*, 2025 (arXiv:2503.19786)