CSCI 5411: Advanced Cloud Architecting
Mid-Term Project Report (Milestone 1)

Meet Maratha (B00994591), Master of Applied Computer Science, Dalhousie University

Contents

List of Figures

1. Project Overview

1.1. AI/GenAI Application Domain Selection and Justification

Zig is an emerging low-level programming language [1] gaining rapid adoption due to its emphasis on safety, performance, and explicit control over hardware. However, developers face significant challenges when learning Zig: its documentation is scarce, lacks structured tutorials, and general-purpose AI coding assistants such as GPT [2], Claude [3], and DeepSeek [4] often provide incorrect or incomplete answers due to limited Zig-specific training data.

To address this gap, this project proposes a Retrieval-Augmented Generation (RAG)-powered AI assistant tailored specifically for Zig. Unlike generalized models, this tool will focus exclusively on Zig's official documentation as its knowledge base. It will try to provide developers with code examples and explanations grounded in it's verified sources. It will also help developers reduce the time spent searching through documentation by delivering context-aware answers to their queries.

1.2. Problem Statement

New Zig developers tend to struggle in finding precise answers to syntax and best-practice to follow while writing their code in Zig's unstructured documentation. This makes them ask those queries to a general purpose AI assistant and trust its solutions, which often lack citations or hallucinate Zig-specific patterns.

This project solves these issues by building a domain-specific AI assistant that retrieves answers directly from Zig's official documentation using RAG to generate responses with code citations such as links to relevant documentation sections. This tool plans to act as a "learning accelerator" for new and upcoming developers that are planning to adopt Zig programming language.

1.3. Business Case

This solutions will also provide a benefit to the enterprises. Adopting Zig as their programming language can help organizations build safer, more efficient low-level systems. However, for the developers working at these organizations the learning curve of Zig is too high. This AI assistant provides three key business benefits to such developers and organizations. Firstly, it increases the developers productivity, reducing time spent debugging incorrect/incomplete AI suggestions. It will also improve code safety, encouraging adoption of Zig by simplifying access to its safety-centric features such as compile-time memory management. Lastly, it will provide all the citations used in generation of a developers answer, allowing then to verify solutions against official documentation, reducing risks in critical systems.

2. Functional Requirements

The functional requirements center on two core AWS services, Amazon SageMaker [5] for hosting the Large Language Model (LLM) inference endpoint and Embedding endpoint and AWS Lambda [6] for orchestrating the Retrieval-Augmented Generation (RAG) pipeline. Lambda functions interact via API Gateway [7] to securely route requests between components. A vector database stores embedded representations of Zig's official documentation, enabling efficient retrieval during the RAG process. Upon receiving a user query, Lambda processes the request, retrieves relevant documentation snippets from the vector database, augments the query with this context, and forwards it to the SageMaker-hosted LLM. The final response—generated by the LLM—is returned to the user through this serverless workflow, ensuring accuracy grounded in Zig's documentation.

3. Non-Functional Requirements

3.1. Security

To mitigate risks, Lambda functions are deployed within a Virtual Private Connection (VPC) and security group to isolate network traffic, allowing communication only with API Gateway and blocking external access. While the Learner Lab restricts Identity Access Management (IAM) customization (using the default role), a real-world deployment would enforce least-privilege IAM roles for each Lambda function. Cross-Origin Resource Sharing (CORS) rules restrict API end-points to pre-approved origins, and AWS Web Access Firewall [8] safeguards the front-end against Distributed Denial-of-Service (DDoS) and SQL injection attacks.

3.2. Scalability

The serverless architecture ensures horizontal scalability: Lambda and API Gateway auto-scale with user demand, while SageMaker endpoints dynamically adjust compute instances to handle concurrent users. Costs are minimized during idle periods due to pay-per-use billing.

3.3. Additional Non-Functional Requirements

The system targets a maximum response latency of 10 seconds per query, achieved through optimized RAG retrieval (vector DB indexing) and SageMaker endpoint tuning. High availability is ensured via Lambda's managed runtime, eliminating downtime even during maintenance or traffic spikes.

4. Architecture Design

4.1. Initial Architecture Diagram

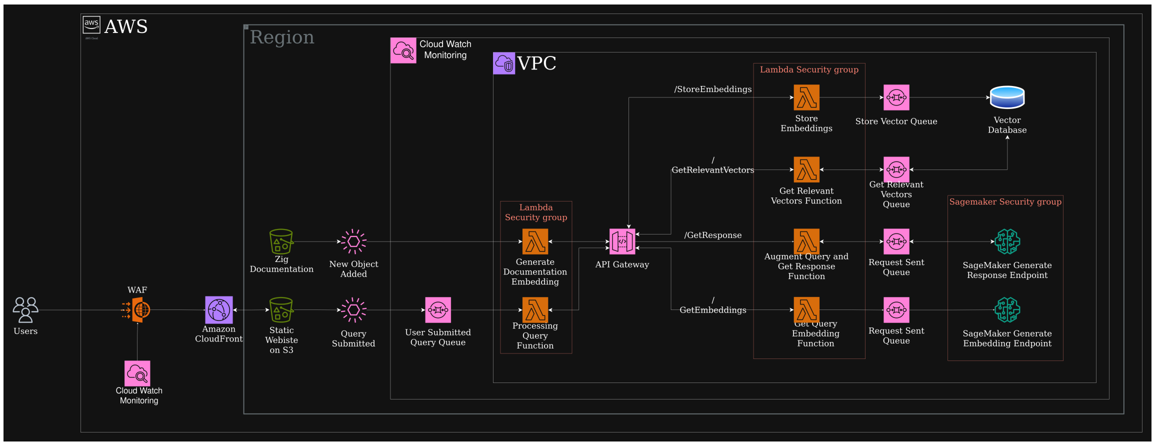

Fig. 1. Initial Architecture Diagram

The architecture comprises of two pipelines namely, document processing and user query handling. In the first pipeline, the Zig programming language documentation is stored in a private S3 bucket. When an object is uploaded to this bucket it triggers a VPC-hosted Lambda function. This Lambda chunks the text and invokes an API Gateway endpoint */GetEmbeddings* to send chunks to a SageMaker-hosted embedding model via a secondary Lambda and Amazon Simple Queue Service (SQS) [9] for reliability. The embedded vectors are then stored in the vector database via the */StoreEmbeddings* endpoint, using another Lambda and Amazon SQS to buffer requests for reliability during SageMaker endpoint downtime.

The user-facing pipeline begins with a CloudFront-distributed [10] static website hosted on S3 and protected by AWS WAF. Queries submitted here trigger a Lambda that first embeds the query via $/GetEmbeddings$ endpoint, then retrieves relevant vectors via $/GetRelevantVectors$ endpoint that uses a Lambda to query the database with SQS for reliability, and finally generates responses via $/GetResponse$ endpoint, which routes the augmented query to a SageMaker-hosted LLM through another Lambda and SQS. All Lambdas share identical security configurations of VPC isolation, security groups restricting traffic to HTTP/HTTPS, and the API Gateway endpoints are secured with CORS rules allowing only specific Lambda function to send requests based on requirements. Finally, the responses are displayed on the front-end alongside cited documentation snippets to the user.

All the lambda functions are also monitored for number of requests handled and the average request time using Amazon CloudWatch [11] along with triggers to send an email when we send a large amount of requests in a small amount of time. This might mean that we are being attacked using DDoS or DoS, so we can protect ourself in that case.

4.2. Data Sequence Diagram

The system employs two distinct data flow pipelines, each visualized in separate sequence diagrams: data ingestion (Figure 2) and data generation (Figure 3).

4.2.1. Data Ingestion Pipeline

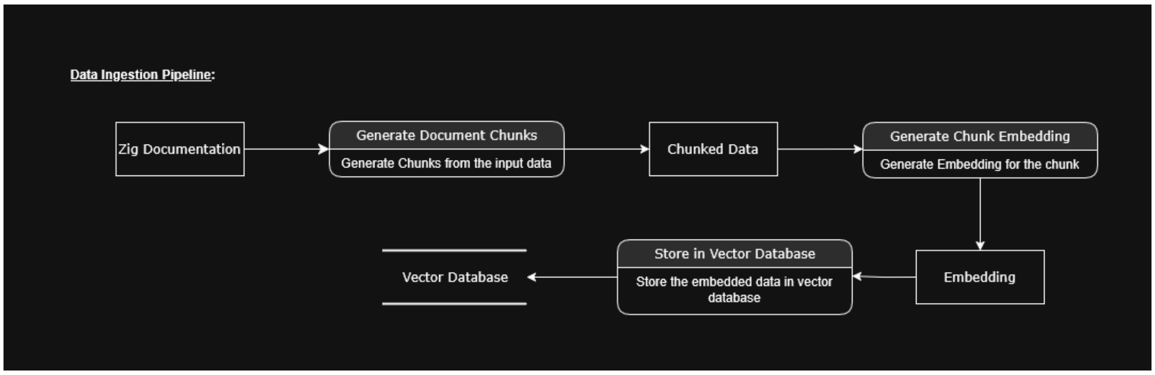

Fig. 2. Data Sequence Diagram for Data Ingestion

This pipeline processes Zig's raw documentation into retrievable embeddings. First, the ”Generate Document Chunks” module splits the raw text into manageable segments. These chunks are forwarded to the ”Generate Chunk Embedding” module, which converts each segment into a numerical embedding vector using a pre-trained model. Finally, the ”Store in Vector Database” module persists these embeddings in the vector database, completing the ingestion cycle.

4.2.2. Data Generation Pipeline

When a user submits a query, the ”Generate Embedding” module converts the input text into an embedding vector. This vector is passed to the ”Get Relevant Vectors” module, which queries the vector database for semantically similar documentation snippets. The retrieved vectors and original query are then fed into the ”Augment Query” module, which synthesizes a context-rich

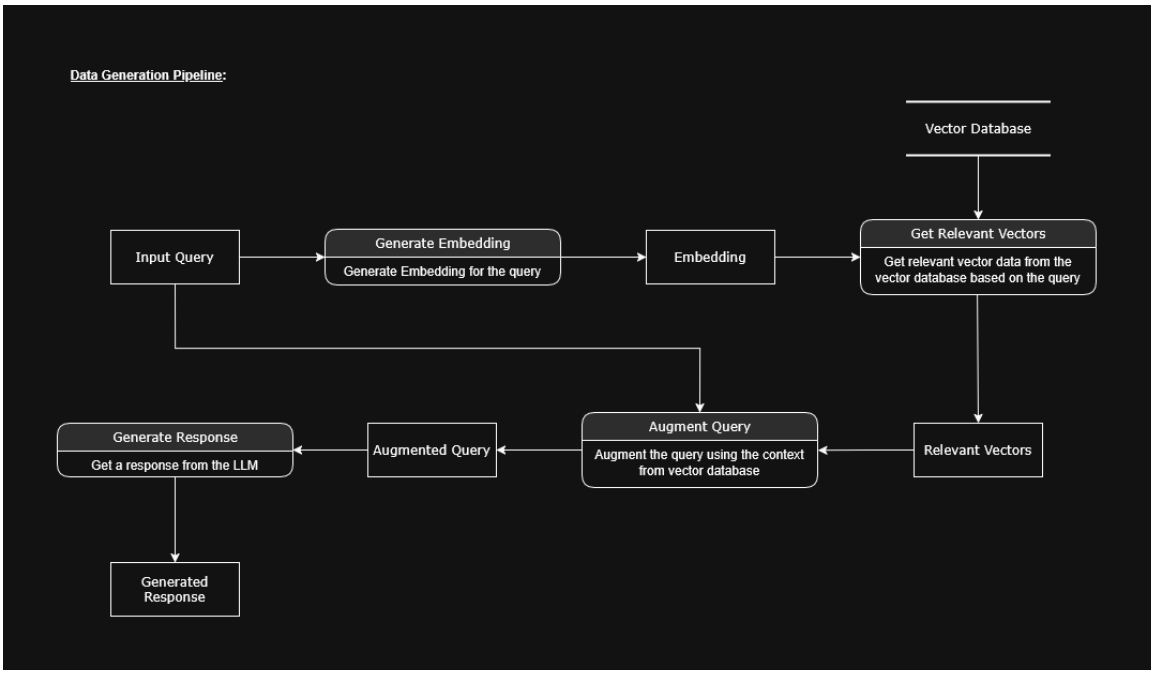
Data Generation Pipeline:
Vector Database
Input Query
Generate Embedding
Generate Embedding for the query
Embedding
Get Relevant Vectors
Get relevant vector data from the vector database based on the query
Generate Response
Get a response from the LLM
Augmented Query
Augment Query
Augment the query using the context from vector database
Relevant Vectors
Generated Response

Fig. 3. Data Sequence Diagram for Data Generation

prompt. This augmented prompt is processed by the ”Generate Response” module, which leverages the LLM to produce a final answer grounded in Zig's official documentation.

5. AWS Services and Tech Stack

The application leverages AWS services for scalability, security, and managed infrastructure. Amazon S3 hosts the static website and stores raw Zig documentation. AWS Lambda (configured within a VPC and security group) executes serverless workflows, including document chunking, embedding generation via SageMaker endpoints, and vector database interactions. Amazon SageMaker hosts both the embedding model and the Large Language Model (LLM) for inference. API Gateway routes requests between Lambda functions securely, while Amazon SQS ensures reliability during transient SageMaker endpoint failures. Network security is enforced via VPCs, security groups (restricting traffic to HTTP/HTTPS), and AWS WAF (protecting against DDoS/SQL injection). Amazon CloudFront accelerates global content delivery for the S3-hosted front-end, and Amazon CloudWatch monitors logs and triggers alerts for anomalies.

The tech stack prioritizes simplicity and interoperability by using Python to power all the Lambda functions for e.g., chunking logic, embedding generation, and RAG orchestration. I use HTML/CSS to build the lightweight, S3-hosted chat interface. I use Chroma to serve as the open-source vector database, integrated with AWS via Lambda to store and retrieve document embeddings.

6. Potential Architectural Challenges

Optimizing Amazon SageMaker instance usage poses a key challenge, as selecting undersized instances for the endpoints may bottleneck embedding generation or LLM inference during peak loads. While Chroma was chosen for its tutorial compatibility and open-source flexibility, its scalability and cost-efficiency in production remain unproven; testing may reveal limitations requiring migration to managed services like Amazon OpenSearch. Additionally, Lambda cold starts—delays during idle periods—could degrade user experience, particularly for infrequently accessed functions. Though Python's ease of use justifies its adoption for initial development, performance-critical components may later require rewrites in compiled languages such as Go.

7. Initial Cost Estimation

The primary costs stem from SageMaker endpoints, though the free tier's 4,000 monthly inferences offset early-stage expenses it can be major contributor in the future. Lambda usage will likely stay within the free tier of 1 Million monthly requests during testing and starting phases of the application deployment, and cost a small fee in the future. S3 storage and static hosting are negligible, but CloudFront charges for data transfer and HTTP requests could escalate with global traffic which can mitigated at the start by restricting users to specific regions, for example, Canada. AWS WAF incurs fixed costs of $5/month/policy and $1/month/rule, which I think is justified for protection against cyber threats. To offset the cost of a managed vector database we can host Chroma on an EC2 free-tier instance which can help us avoid database licensing fees, but it will limit compute capacity, which can necessitate upgrades in future as data and usage grows. Long-term scalability will require revisiting pricing models for the application to align costs with user demand.

References

[1] Zig, “Zig documentation,” 2025, accessed: 2025-05-17. [Online]. Available: https://ziglang.org/documentation/master/
[2] OpenAI, “Chatgpt,” 2025, accessed: 2025-05-17. [Online]. Available: https://chat.openai.com/chat
[3] Anthropic, “Claude,” 2025, accessed: 2025-05-17. [Online]. Available: https://claude.ai/
[4] DeepSeek-AI, D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, X. Zhang, X. Yu, Y. Wu, Z. F. Wu, Z. Gou, Z. Shao, Z. Li, Z. Gao, A. Liu, B. Xue, B. Wang, B. Wu, B. Feng, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan, D. Dai, D. Chen, D. Ji, E. Li, F. Lin, F. Dai, F. Luo, G. Hao, G. Chen, G. Li, H. Zhang, H. Bao, H. Xu, H. Wang, H. Ding, H. Xin, H. Gao, H. Qu, H. Li, J. Guo, J. Li, J. Wang, J. Chen, J. Yuan, J. Qiu, J. Li, J. L. Cai, J. Ni, J. Liang, J. Chen, K. Dong, K. Hu, K. Gao, K. Guan, K. Huang, K. Yu, L. Wang, L. Zhang, L. Zhao, L. Wang, L. Zhang, L. Xu, L. Xia, M. Zhang, M. Zhang, M. Tang, M. Li, M. Wang, M. Li, N. Tian, P. Huang, P. Zhang, Q. Wang, Q. Chen, Q. Du, R. Ge, R. Zhang, R. Pan, R. Wang, R. J. Chen, R. L. Jin, R. Chen, S. Lu, S. Zhou, S. Chen, S. Ye, S. Wang, S. Yu, S. Zhou, S. Pan, S. S. Li, S. Zhou, S. Wu, S. Ye, T. Yun, T. Pei, T. Sun, T. Wang, W. Zeng, W. Zhao, W. Liu, W. Liang, W. Gao, W. Yu, W. Zhang, W. L. Xiao, W. An, X. Liu, X. Wang, X. Chen, X. Nie, X. Cheng, X. Liu, X. Xie, X. Liu, X. Yang, X. Li, X. Su, X. Lin, X. Q. Li, X. Jin, X. Shen, X. Chen, X. Sun, X. Wang, X. Song, X. Zhou, X. Wang, X. Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Y. Zhang, Y. Xu, Y. Li, Y. Zhao, Y. Sun, Y. Wang, Y. Yu, Y. Zhang, Y. Shi, Y. Xiong, Y. He, Y. Piao, Y. Wang, Y. Tan, Y. Ma, Y. Liu, Y. Guo, Y. Ou, Y. Wang, Y. Gong, Y. Zou, Y. He, Y. Xiong, Y. Luo, Y. You, Y. Liu, Y. Zhou, Y. X. Zhu, Y. Xu, Y. Huang, Y. Li, Y. Zheng, Y. Zhu, Y. Ma, Y. Tang, Y. Zha, Y. Yan, Z. Z. Ren, Z. Ren, Z. Sha, Z. Fu, Z. Xu, Z. Xie, Z. Zhang, Z. Hao, Z. Ma, Z. Yan, Z. Wu, Z. Gu, Z. Zhu, Z. Liu, Z. Li, Z. Xie, Z. Song, Z. Pan, Z. Huang, Z. Xu, Z. Zhang, and Z. Zhang, “Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning,” 2025. [Online]. Available: https://arxiv.org/abs/2501.12948
[5] Amazon, “Sagemaker,” 2025, accessed: 2025-05-17. [Online]. Available: https://aws.amazon.com/sagemaker/
[6] Amazon Web Services, “Lambda function,” 2025, accessed: 2025-05-17. [Online]. Available: https://aws.amazon.com/lambda/
[7] ——, “Api gateway,” 2025, accessed: 2025-05-17. [Online]. Available: https://docs.aws.amazon.com/apigateway/
[8] ——, “Web access firewall,” 2025, accessed: 2025-05-17. [Online]. Available: https://aws.amazon.com/waf/
[9] Amazon, “Simple queue service,” 2025, accessed: 2025-05-17. [Online]. Available: https://docs.aws.amazon.com/sqs/
[10] ——, “Cloudfront,” 2025, accessed: 2025-05-17. [Online]. Available: https://docs.aws.amazon.com/cloudfront/
[11] ——, “Cloudwatch,” 2025, accessed: 2025-05-17. [Online]. Available: https://docs.aws.amazon.com/cloudwatch/