

Capstone Report

Renesas Technical Knowledge Base RAG System

1. Introduction

This project implements a Retrieval-Augmented Generation (RAG) system specifically designed for Renesas Electronics' technical knowledge base. The system addresses the challenge of efficiently accessing and retrieving information from vast repositories of technical documentation on microcontrollers, development tools, and hardware components.

Renesas Electronics provides extensive technical documentation for their products, but searching for specific information can be challenging due to the volume and complexity of the content. This solution leverages modern NLP techniques and cloud infrastructure to create an intelligent assistant that can understand technical queries and provide precise, relevant answers by retrieving and synthesizing information from the knowledge base.

The system combines web crawling, content extraction, natural language processing, and a user-friendly interface to provide a comprehensive solution for technical support and information retrieval. By implementing RAG architecture, the system ensures responses are grounded in actual documentation rather than generating potentially inaccurate information.

2. Existing Solutions in Market and Their Limitations

Current Market Solutions

1. Traditional Knowledge Base Search Systems

- Basic keyword-based search engines embedded in documentation portals
- Simple FAQ systems with predefined answers to common questions

2. General-Purpose Chatbots

- Generic AI assistants with limited domain knowledge
- Not specifically trained on technical microcontroller documentation

3. Document Retrieval Systems

- Systems that retrieve whole documents rather than specific answers

- Require users to parse through lengthy technical content

Limitations of Existing Solutions

1. Lack of Domain Specificity

- General-purpose systems struggle with specialized technical terminology
- Poor understanding of relationships between components, features, and specifications

2. Inability to Process Complex Queries

- Simple keyword search fails to understand context or intent
- Cannot handle multi-part technical questions

3. Poor Information Synthesis

- Most systems retrieve documents but don't synthesize answers
- Users must still manually extract relevant information

4. Limited Context Understanding

- Inability to maintain context across a conversation
- Each query treated independently without building on previous exchanges

5. Absence of Visual Content Processing

- Most systems ignore diagrams, schematics, and other visual content
- Technical documentation often relies heavily on visual elements

3. Our Approach

Our solution implements a comprehensive RAG (Retrieval-Augmented Generation) system specifically tailored for Renesas Electronics' technical content. The approach consists of three primary components:

Data Collection and Processing Pipeline

1. Hierarchical Web Crawling

- Two-stage crawling approach: first mapping the site structure, then extracting content
- Preservation of category relationships and technical context

2. Comprehensive Content Extraction

- Extraction of text, images, tables, PDFs, and other technical content
- Maintenance of relationships between different content types

3. Structured Data Organization

- Organization of content into a clean, navigable directory structure
- Preservation of metadata and relationships

Knowledge Embedding and Retrieval

1. Multimodal Embedding Generation

- Text embeddings for technical documentation
- Image embeddings for diagrams, screenshots, and visual content
- Combined vector representation of content

2. AWS Bedrock Integration

- Leveraging AWS Bedrock services for vector storage and search
- Optimized retrieval based on semantic similarity

3. Contextual Retrieval Mechanisms

- Smart retrieval based on query intent and content type
- Retrieval of appropriate information sources (text, images, tables)

Intelligent Response Generation

1. Query Understanding

- Analysis of technical queries to determine intent and required information
- Breaking down complex queries into retrievable components

2. Source-Grounded Response Generation

- Generation of responses strictly based on retrieved information
- Citation of sources for transparency

3. User Interface Integration

- Streamlit-based interface for easy interaction
- Display of both answers and supporting evidence

This approach ensures the system provides accurate, relevant responses to technical queries while maintaining transparency about information sources.

4. Data Sources Used

Our system uses the following data sources:

Primary Data Source

- **Renesas Electronics Knowledge Base:** Comprehensive technical documentation from Renesas Electronics' support portal (en-support.renesas.com/knowledgeBase)

Content Types Extracted

1. Technical Articles

- Detailed explanations of hardware components and features
- Step-by-step guides for development environments
- Troubleshooting information and common solutions

2. Product Documentation

- Microcontroller specifications for different product families (RH850, RA, RX, RL78)
- Development board information and usage guides
- Software and firmware documentation

3. Visual Content

- Screenshots of development environments and tools
- Circuit diagrams and hardware schematics
- User interface illustrations

4. Technical Specifications

- Detailed tables of electrical specifications
- Performance metrics and compatibility information
- Configuration parameters

5. Code Examples

- Programming examples for various microcontrollers

- Sample configurations and implementations
- API usage demonstrations

The data covers multiple product families including:

- RH850 Family (automotive microcontrollers)
- RA Family (Arm® Cortex®-based MCUs)
- RX Family (32-bit microcontrollers)
- RL78 Family (low-power microcontrollers)
- RZ Family (MPUs)

The system processes and indexes approximately 152 articles with images and over 500 technical documents, creating a comprehensive knowledge base for Renesas products.

5. Explanation of the Code and System Design

System Architecture

The system architecture follows a modular design with three main components:

1. Data Collection (layoutCrawl)

- Maps website structure and relationships
- Identifies categories, subcategories, and content URLs
- Outputs structured data for content extraction

2. Content Processing (articleCrawl)

- Extracts content based on site structure mapping
- Processes text, images, tables, and other elements
- Organizes content into structured directories

3. RAG Implementation (RAG)

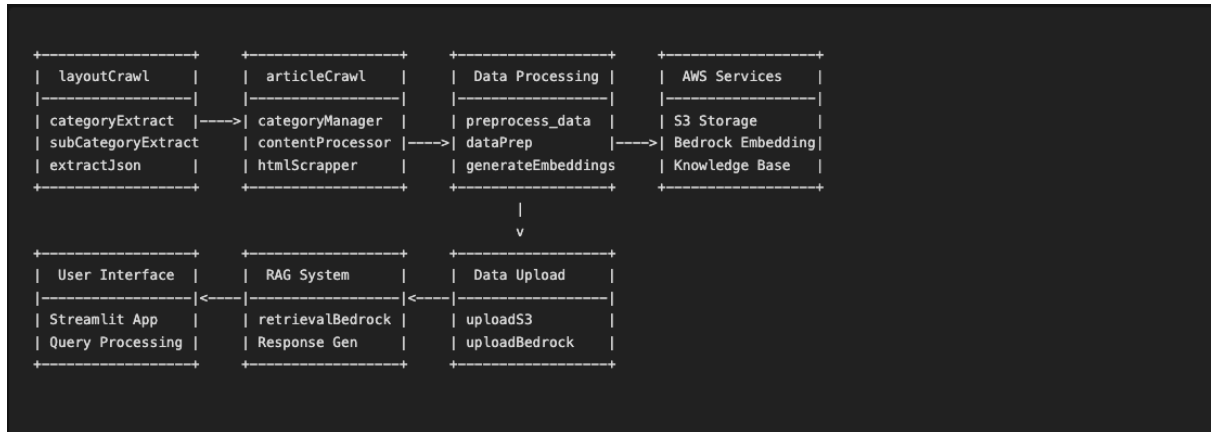
- Processes data for embedding generation
- Implements AWS Bedrock integration
- Handles search and retrieval functionality

4. User Interface

- Streamlit-based web interface for user interaction

- Displays responses with supporting evidence
- Provides intuitive query input

Block Diagram



Key Code Components

1. Web Crawling and Content Extraction

```
# Sample from categoryExtract.py
def extract_categories_and_subcategories(driver, logger):
    categories = []
    try:
        # Wait for Angular to fully load the content
        WebDriverWait(driver, 20).until(
            EC.presence_of_element_located((By.CSS_SELECTOR, "div.panel
l.panel-default"))
        )

        # Find all category sections that actually contain content
        category_sections = driver.find_elements(By.CSS_SELECTOR,
            "div.panel.panel-default[style*='display: block']")

        # Extract categories and subcategories
        for section_index, category_section in enumerate(category_section
s, 1):
            # Implementation details...
```

2. Content Processing and Embedding Generation

```

# Sample from generateEmbeddings.py
def generate_titan_embedding(client, text, model_id):
    """Generate embedding for a text using AWS Bedrock Titan Embedding model."""
    if not text:
        return None

    try:
        # Prepare the request body for the model
        request_body = json.dumps({
            "inputText": text
        })

        # Call the Bedrock runtime
        response = client.invoke_model(
            modelId=model_id,
            body=request_body
        )

        # Process the response
        response_body = json.loads(response.get('body').read())
        embedding = response_body.get('embedding')

        return embedding
    except Exception as e:
        print(f"Error generating embedding: {str(e)}")
        return None

```

3. Retrieval and Response Generation

```

# Sample from retrievalBedrock.py
def retrieve_from_kb(query, top_k=5, score_threshold=0.6):
    """Retrieve relevant content from Knowledge Base based on query."""
    try:
        # Retrieval using text query
        response = bedrock_agent.retrieve(
            knowledgeBaseId=kb_id,

```

```

    retrievalQuery={
        "text": query
    },
    retrievalConfiguration={
        "vectorSearchConfiguration": {
            "numberOfResults": top_k
        }
    }
)

# Process retrieved results
retrieved_results = response.get("retrievalResults", [])

# Filter by score threshold and sort by relevance
# Implementation details...

```

6. Sample Interactions with the System

Example 1: Basic Query about Device File

User Query:

"How do I find RH850 device file?"

System Response:

"The RH850 device file can be found in the install directory, specifically in the path C:\Program Files (x86)\Renesas Electronics\CS\Device\RH850\Devicefile. It is not necessary to select the file manually, as it is done automatically if a device is selected in the CS Device Selection dialogue. If the device is not listed, check that you are using the latest version of CS. The device file is a binary file that describes the device and is used to create header-files for a C-compiler and other files."

Sources Referenced:

1. Title: RH850 find dvf file for CS+
URL: <https://en-support.renesas.com/knowledgeBase/21153486>
Relevance Score: 0.91

Example 2: Technical Implementation Question

User Query:

"How can I configure nested interrupts in RH850?"

System Response:

"To configure nested interrupts in RH850, you need to set the ISPR (Interrupt Status Priority Register) and INTC2.EIC registers. The EIC register needs to be configured with the appropriate values to enable interrupt nesting. You can set different priority levels for interrupts, and higher priority interrupts can preempt lower priority ones. In your code, you'll need to configure the ICR.INTM bit to 0 to allow nesting. Remember to properly handle the context saving and restoring in your interrupt service routines."

Sources Referenced:

1. Title: Nested Interrupt in RH850
URL: <https://en-support.renesas.com/knowledgeBase/19879255>
Relevance Score: 0.88
2. Title: RH850 interrupt configuration
URL: <https://en-support.renesas.com/knowledgeBase/19487624>
Relevance Score: 0.72

Example 3: Image-Related Query**User Query:**

"Show me how to set up debugging with E1 emulator"

System Response:

"To set up debugging with the E1 emulator for Renesas microcontrollers, follow these steps:

1. Connect the E1 emulator to your PC via USB and to your target board through the debug connector
2. Launch CS+ IDE and create or open your project
3. Go to the debug configuration settings by selecting 'Debug' → 'Connection Settings'
4. In the 'Emulator Settings' section, select 'E1' as your emulator
5. Configure the appropriate communication settings (typically LPD 4-Wire for RH850)
6. Set the oscillator frequency to match your target board
7. Click 'Connect' to establish the debugging connection"

Sources Referenced:

1. E1 emulator setup interface screenshot
Relevance Score: 0.85
2. Title: RH850: Programmer change from E1 to E2
Relevance Score: 0.77

7. Limitations

Despite the system's capabilities, several limitations should be acknowledged:

1. Content Coverage Limitations

- Limited to publicly available documentation
- May not include the most recent product updates
- Specialized or proprietary documentation may be missing

2. Technical Constraints

- Handling of complex technical diagrams is limited
- Mathematical equations and specialized notations may not be properly interpreted
- Some PDF content may not be fully extractable

3. Query Understanding Challenges

- Highly specialized technical terminology might be misinterpreted
- Complex multi-part queries with dependencies can be challenging
- Queries requiring complex reasoning across multiple documents may yield incomplete answers

4. RAG System Limitations

- "Hallucination" risk when information is sparse
- Limited ability to infer beyond explicitly stated information
- Context window constraints limit the amount of information that can be processed at once

5. User Experience Considerations

- Response generation time can be slow for complex queries
- Limited interactive capabilities for diagrams and schematics
- No direct integration with development environments

6. Data Freshness

- Knowledge base requires regular updates to remain current
- New product releases may not be immediately reflected
- System has no automatic update mechanism

8. Future Scope

The following enhancements could address current limitations and expand system capabilities:

1. Enhanced Content Processing

- Integration with PDF parsing libraries for better document extraction
- Advanced processing of schematics and technical diagrams
- Support for code snippets with syntax highlighting and execution

2. Expanded Data Sources

- Integration with Renesas GitHub repositories for code examples
- Incorporation of forum discussions and community knowledge
- Addition of application notes and technical white papers

3. Advanced Query Capabilities

- Support for multi-turn debugging sessions
- Code completion and suggestion features
- Natural language to code translation for configuration

4. User Experience Improvements

- Interactive circuit diagrams and schematics
- Integration with development environments like e2 studio
- Mobile application for on-the-go assistance

5. Knowledge Graph Integration

- Building a technical knowledge graph of components and relationships
- Enabling more complex reasoning across documentation
- Visualization of relationships between products and features

6. Multilingual Support

- Extension to support multiple languages for global developer teams
- Localization of technical terms and component names
- Region-specific documentation and compliance information

7. Community Contribution Mechanism

- Allowing users to suggest corrections or additions
- Integration of verified community knowledge
- Rating system for response quality feedback

8. Custom Deployment Options

- On-premises deployment for organizations with security requirements
- Integration with existing enterprise knowledge management systems
- API access for integration with custom tools and workflows

These enhancements would transform the system into a more comprehensive technical assistant for Renesas developers, addressing current limitations while expanding capabilities to meet evolving needs.