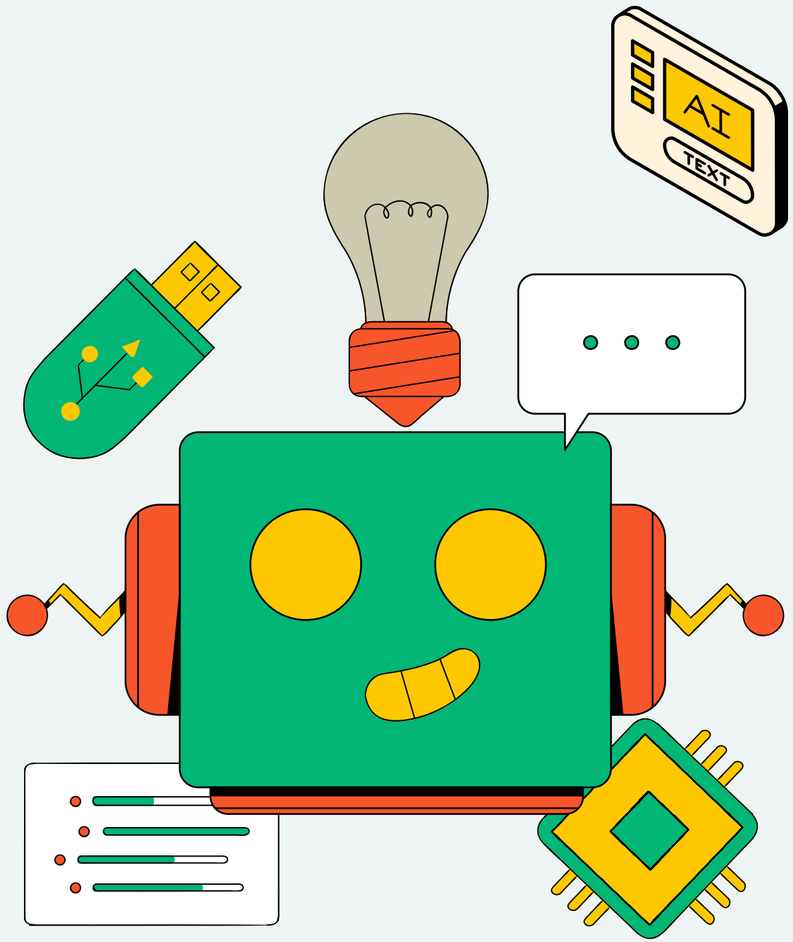
THYNK UNLIMITED

WE LEARN FOR THE FUTURE



CHATBOT USING RAG AND AUTOCRAWL

GROUP 8

NGUYEN THANH LONG – 20214912

NGUYEN MINH CUONG – 20210140

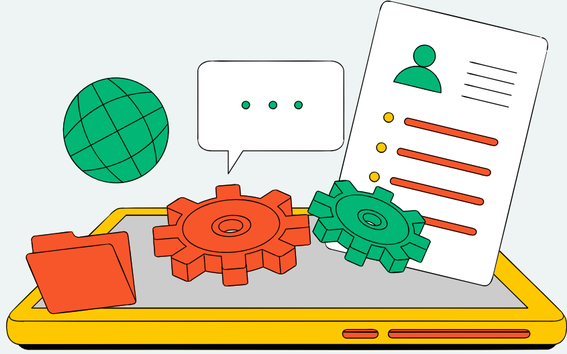
DOAN NGOC CUONG – 20210141

LUU ANH DUC – 20204875

TABLE OF CONTENT

I.INTRODUCTION

1. PROJECT OVERVIEW

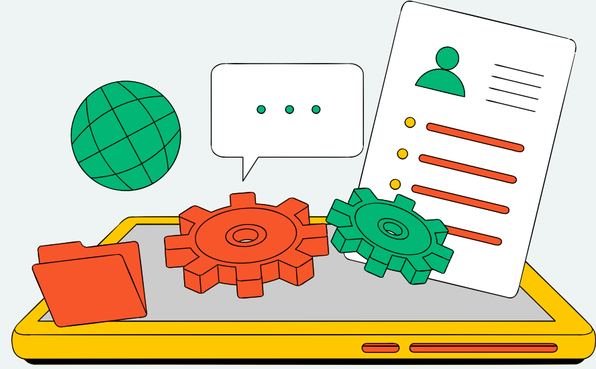


Chatbot Development:

- **Combines Retrieval-Augmented Generation (RAG) and AutoCrawl Technology.**
- **Key Features:**
  - **Data crawling and processing pipelines.**
  - **Retrieval-based architecture for accurate, contextually relevant responses.**
  - **Integration of real-time web crawling with LLM-based synthesis.**
- **Purpose: Provides users with intelligent, real-time answers using dynamic data collection and advanced NLP techniques.**

I.INTRODUCTION

2. MOTIVATION AND OBJECTIVES



Motivation:

- **Rapid information growth demands smarter retrieval tools.**
- **Traditional chatbots are limited by static datasets and outdated knowledge bases.**
- **Need for dynamic, reliable, and context-aware responses.**

Key Objectives:

- **Develop a scalable solution for real-time question answering.**
- **Ensure up-to-date and relevant interactions.**
- **Explore the synergy between AutoCrawl and RAG to enhance AI systems.**

II. METHODOLOGY.

1. System Architecture.

Data Collection and Processing

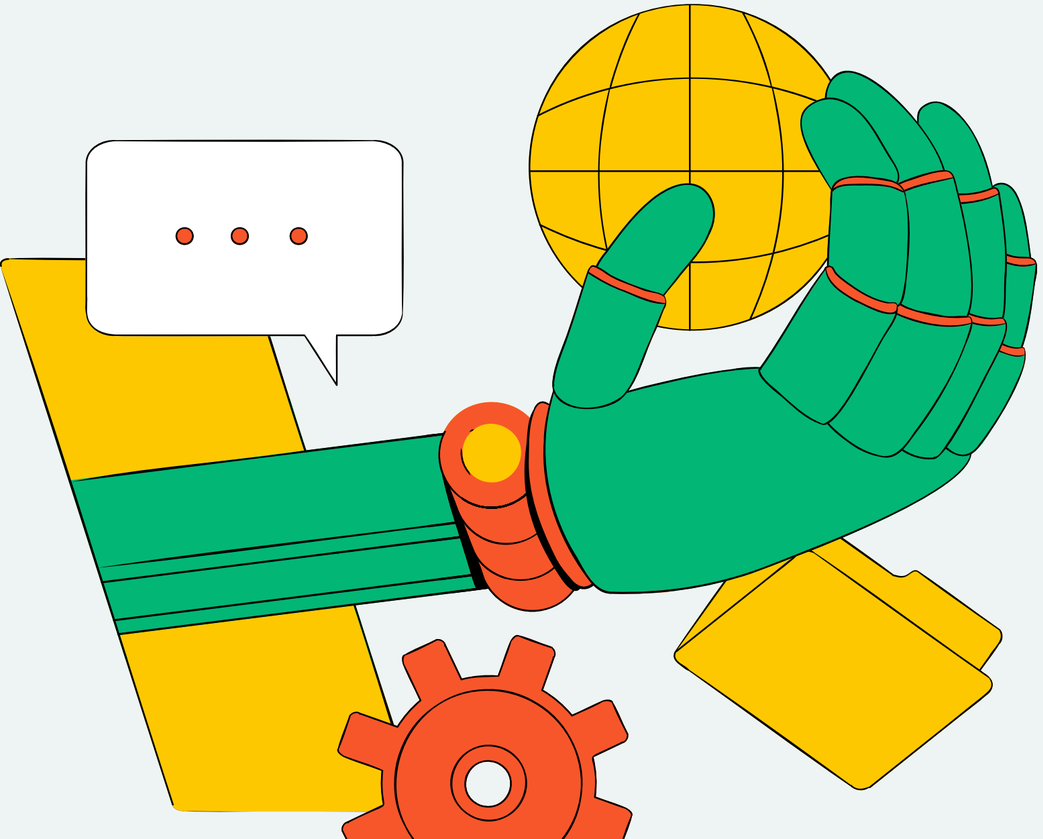
Indexing and Vector Search

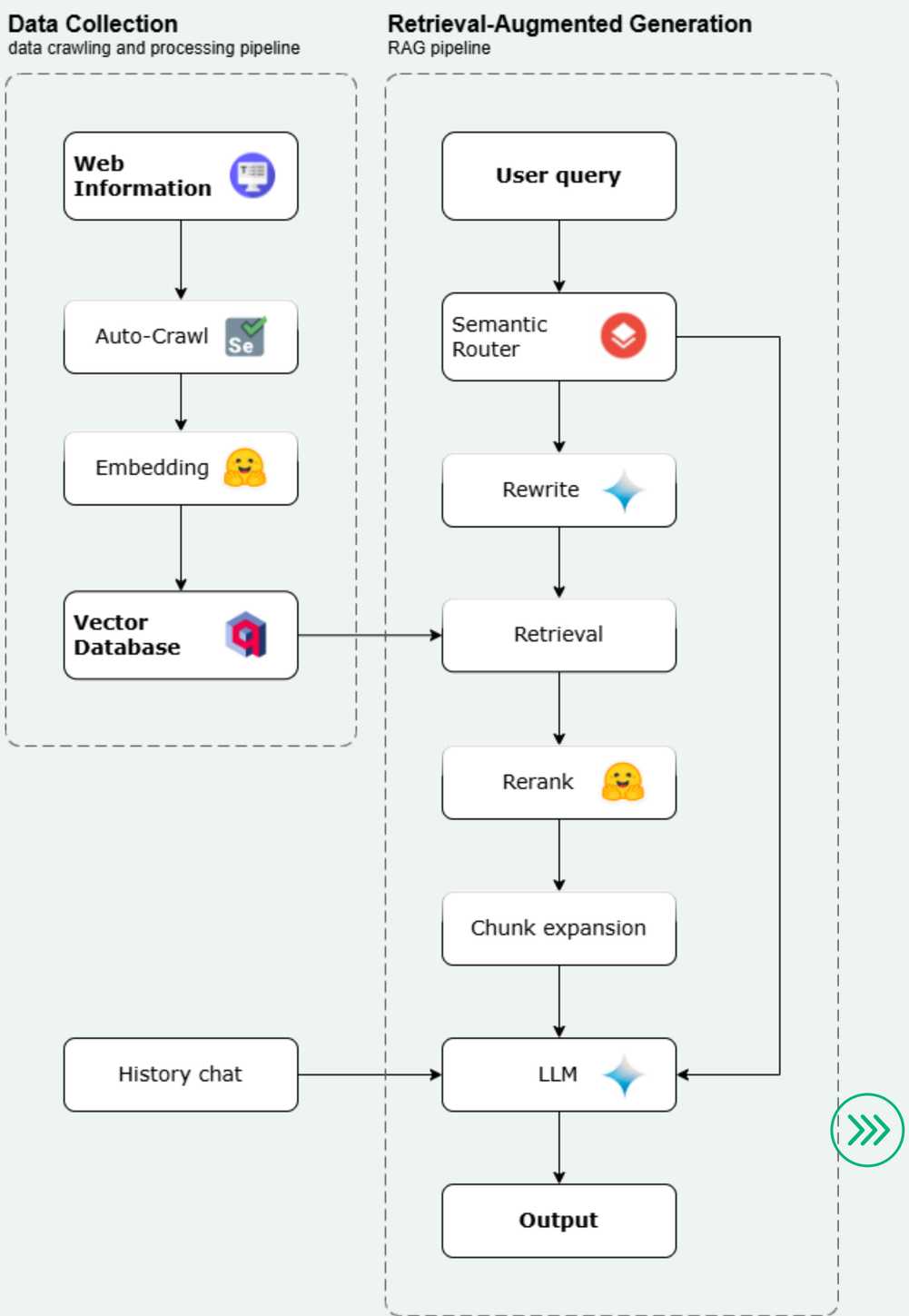
RAG Pipeline

Implementation Details

III. Methodology.

1. System Architecture





III. Methodology.

2. Data Collection and Processing

Data Collection Process:

- URL Validation:
  - Ensures only same-domain URLs are processed using is_valid_url function.
- Content Extraction:
  - Extracts text from key HTML tags (h1, h2, p, article) using BeautifulSoup.
  - Extracts relevant links for further crawling, adhering to user-defined limits.
- Crawling Mechanism:
  - Recursively visits valid URLs and follows links.
  - Continues until the specified depth or link limit is reached.
- Efficiency Measures:
  - Tracks visited URLs to avoid duplicate processing.

III. Methodology.

3. Indexing and Vector Search

Process:

1. Semantic Chunking:
    - Divides data into 150–350 token chunks.
    - Uses a sliding window to maintain semantic coherence.
    - Chunk boundaries defined by:
        - Semantic similarity thresholds.
        - Token length limits.
2. Embedding:
    - Each chunk is embedded into a dense 768-dimension vector using a Sentence Transformer model.
    - Embeddings enable semantic search and retrieval.

Output:

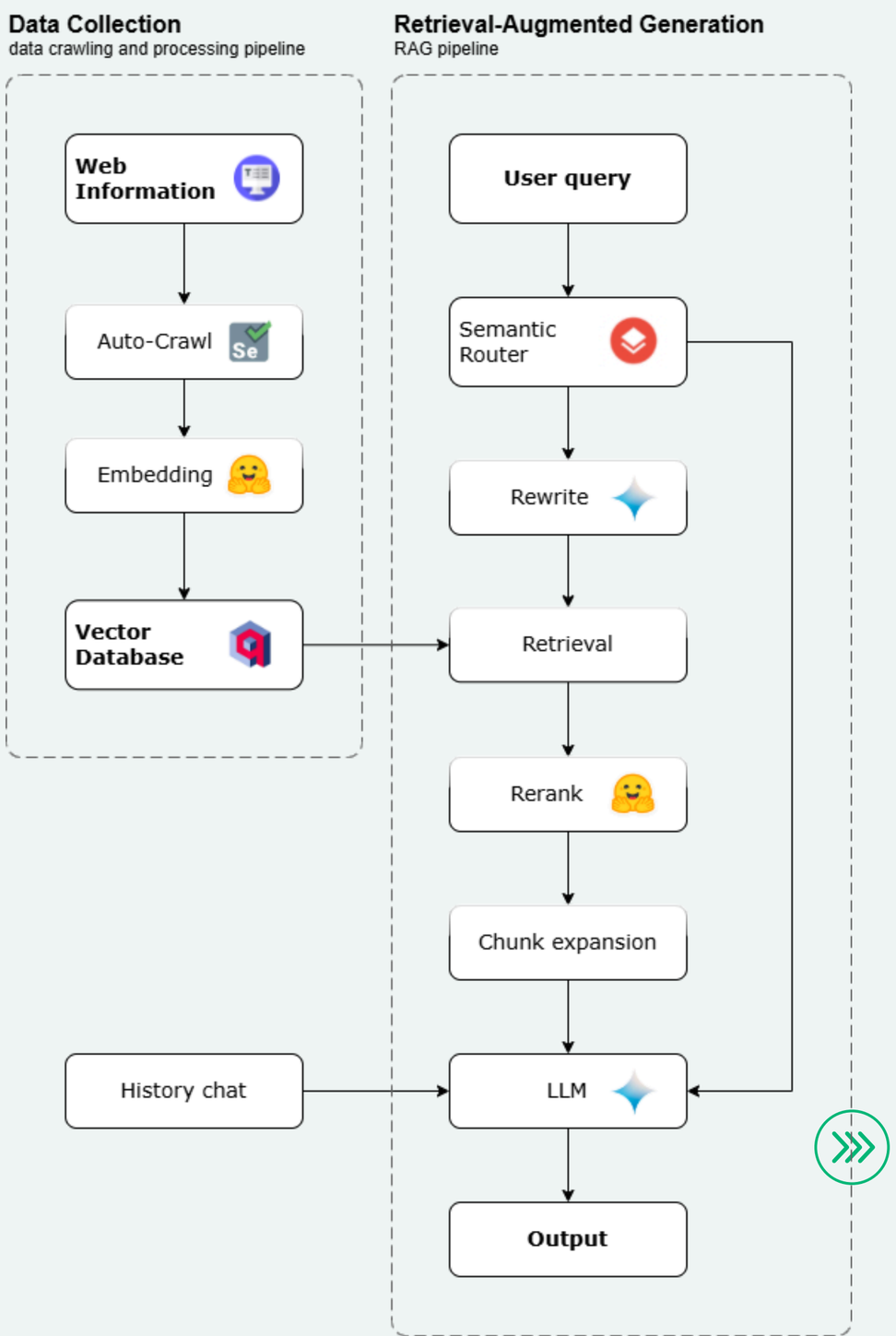
- Semantically coherent chunks saved with references to source documents.
- Forms the foundation for semantic search and retrieval tasks.

III. Methodology.

4. RAG pipeline

The RAG pipeline consists of several key steps to effectively process user queries and generate accurate, context-aware responses:

1. **Semantic Router**
2. **Query Rewrite**
3. **Semantic Search**
4. **Reranking**
5. **Chunk Expansion**
6. **Response Generation**
    -



III. Methodology.

5. Implementation Overview

Three Primary Components:

1. **Backend: Core system managing data collection, processing, querying, and generating intelligent responses.**
2. **Frontend: User interface for interacting with the chatbot.**
3. **Deployment: Ensures scalability, security, and high availability through containerized environments.**

III. Methodology.

5. Implementation details

Backend Implementation

Technologies:

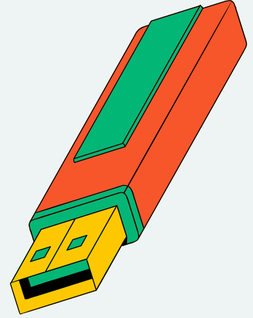
- **Flask: API development.**
- **Qdrant: Vector database for dense embeddings.**
- **Google Gemini API: Query rewriting and response generation.**
- **Beautiful Soup: Web crawling and content extraction.**
- **LangChain: Orchestrates the RAG pipeline.**

Key Features:

1. **Crawling and Content Processing:**
   - **/crawl: Initiates crawling with URL and depth specifications.**
   - **/crawl status: Tracks crawling progress.**
2. **Database Management:**
   - **/save db: Saves processed data into Qdrant collections.**
   - **/get collections: Lists available collections.**
   - **/change collection: Switches active collections.**
3. **Query Handling:**
   - **/send message: Processes user queries using the RAG pipeline.**
4. **Health Check:**
   - **/health: Verifies backend system status.**

III. Methodology.

5. Implementation details



Frontend Implementation

Technologies:

- **JavaScript, HTML, CSS: Core development.**
- **React.js (Optional): Dynamic single-page applications.**
- **Bootstrap: Ensures responsive design.**

Key Features:

1. **Chat Interface:**
   - **Accepts user input and displays responses.**
   - **Supports Markdown for formatted bot responses.**
2. **Dynamic Collection Management:**
   - **Fetches and switches between backend collections.**
3. **Web Crawling:**
   - **Initiates crawling tasks with specified parameters.**
   - **Displays real-time progress updates.**
4. **Error Handling:**
   - **Provides detailed feedback for invalid inputs or backend issues.**

III. Methodology.

5. Implementation details

Deployment

Technologies:

- **Docker: Containerization of backend and frontend.**
- **Docker Compose: Multi-container orchestration.**
- **Cloud Platforms: AWS, Google Cloud, or Azure for hosting.**

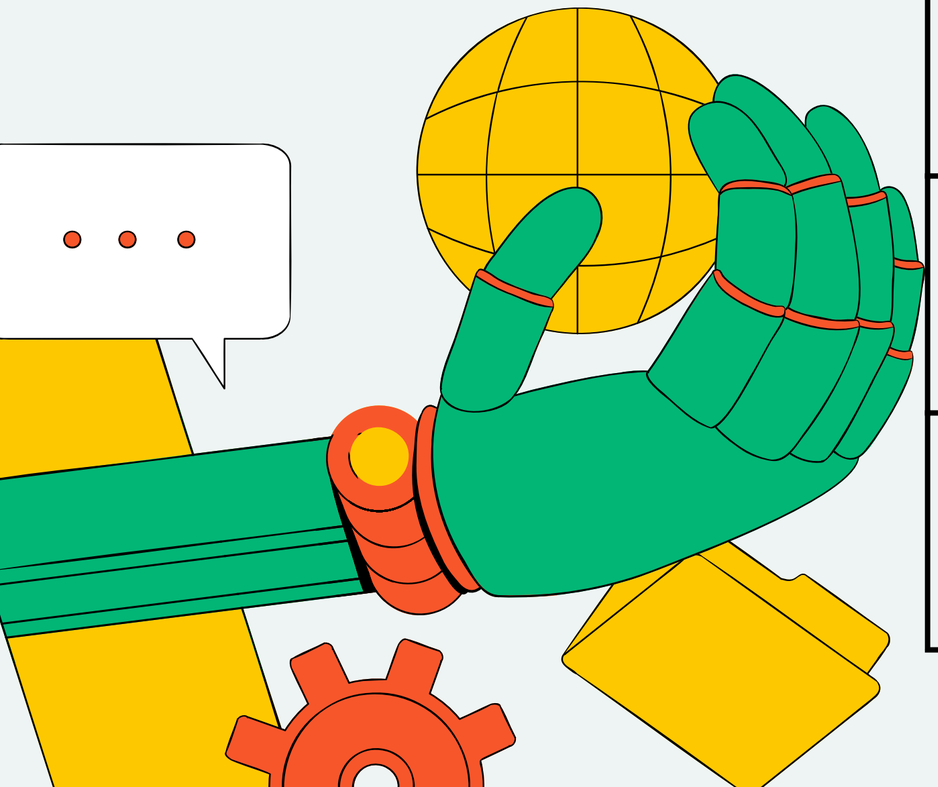
Deployment Steps:

1. **Backend:**
   - **Containerized Flask app with Qdrant integration.**
   - **Secure API keys using .env files.**
2. **Frontend:**
   - **Static files hosted on Netlify or Vercel.**
   - **HTTPS-enabled for secure user interactions.**
3. **Networking and Security:**
   - **Enforced HTTPS for all communications.**
   - **Secure API key handling.**

Scalability:

- **Horizontal scaling of backend and Qdrant containers.**
- **Caching frequently accessed data to reduce API calls.**

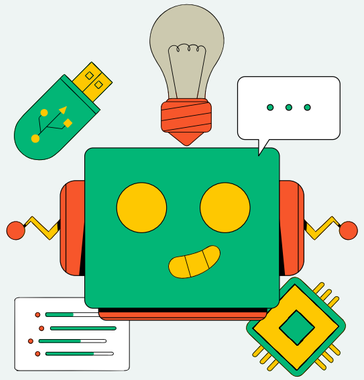
IV.EXPERIMENT RESULTS



Link	Number of questions	Correct crawls	Correct answers
https://cmccloud.vn/	30	24	24
https://cmctelecom.vn/	20	12	10
Total	50	36	34

V.CONCLUSION AND FUTURE WORKS

Conclusion



The project successfully implemented a Retrieval-Augmented Generation (RAG) system integrated with an AutoCrawl pipeline to address dynamic information retrieval and response generation. Key accomplishments of the project include:

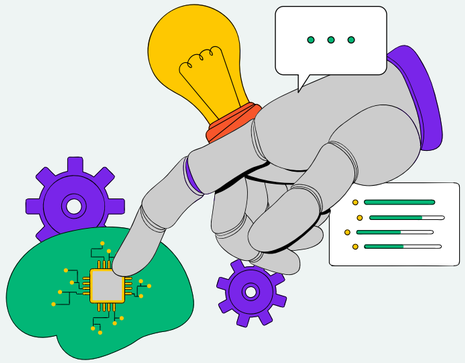
- **Efficient Data Collection:** The AutoCrawl pipeline automated the collection and embedding of relevant information into a vector database.

- **Effective Retrieval and Response Generation:** The combination of semantic search, multiple modules such as rewrite, rerank, and response synthesis using the Google Gemini API provided accurate and context-aware responses to user queries.

- **Scalability:** The modular architecture supports the integration of future enhancements, such as sparse search and knowledge graph-based retrieval.

V.CONCLUSION AND FUTURE WORKS

Future works



While the current RAG system demonstrates promising results, several areas have been identified for further development and improvement. These enhancements focus on refining existing modules and exploring additional functionalities to optimize the system's performance.

- **Enhancing Data Filtering in AutoCrawl:** Improve the data collection process by implementing advanced filtering techniques to exclude irrelevant or low-quality content, ensuring higher precision in retrieval results.

- **Rerank Model Fine-Tuning:** One observed challenge is that the rerank model, namdp-ptit/ViRanker, occasionally performs suboptimally, particularly in specific contexts or query types.

- **Additional Modules – Query Decomposition:** Finally, expanding the system's capabilities by integrating additional modules such as Query Decomposition could offer further enhancements.

THANKS FOR LISTENING