processes

MDPI

Article

Domain-Specific Manufacturing Analytics Framework: An Integrated Architecture with Retrieval-Augmented Generation and Ollama-Based Models for Manufacturing Execution Systems Environments

Hangseo Choi and Jongpil Jeong *

Department of Smart Factory Convergence, Sungkyunkwan University, 2066 Seobu-ro, Jangan-gu, Suwon 16419, Gyeonggi-do, Republic of Korea; choinara@g.skku.edu
* Correspondence: jpjeong@skku.edu

Abstract: To support data-driven decision-making in a Manufacturing Execution System (MES) environment, a system that can quickly and accurately analyze a wide range of production, quality, asset, and material information must be deployed. However, existing MES data management approaches rely on predefined queries or report templates that lack flexibility and limit real-time decision support. In this paper, we proposes a domain-specific Retrieval-Augmented Generation (RAG) architecture that extends LangChain's capabilities with Manufacturing Execution System (MES)-specific components and the Ollama-based Local Large Language Model (LLM). The proposed architecture addresses unique MES requirements including real-time sensor data processing, complex manufacturing workflows, and domain-specific knowledge integration. It implements a three-layer structure: an application layer using FastAPI for high-performance asynchronous processing, an LLM layer for natural language understanding, and a data storage layer combining MariaDB, Redis, and Weaviate for efficient data management. The system effectively handles MES-specific challenges such as schema relationships, temporal data processing, and security concerns without exposing sensitive factory data. This is an industry-specific, customized approach focusing on problem-solving in manufacturing sites, going beyond simple text-based RAG. The proposed architecture considers the specificity of data sources, real-time and high-availability requirements, the reflection of domain knowledge and workflows, compliance with security and quality control regulations, and direct interoperability with MES systems. The architecture can be further enhanced through integration with various manufacturing systems, an advanced LLM, and distributed processing frameworks while maintaining its core focus on MES domain specialization.

Keywords: manufacturing execution system (MES); retrieval-augmented generation (RAG); MES domain-specific RAG; local large language model (LLM); real-time data processing; FastAPI

check for updates

Academic Editor: Abraham Kabutey

Received: 24 December 2024
Revised: 17 February 2025
Accepted: 24 February 2025
Published: 27 February 2025

Citation: Choi, H.; Jeong, J. Domain-Specific Manufacturing Analytics Framework: An Integrated Architecture with Retrieval-Augmented Generation and Ollama-Based Models for Manufacturing Execution Systems Environments. *Processes* **2025**, *13*, 670. https://doi.org/10.3390/pr13030670

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

The large volumes of data generated on the manufacturing floor are a key resource for a variety of decision-making processes, including real-time monitoring, quality improvement, and productivity enhancement [1]. Manufacturing data can be broadly classified as facility or MES data. Facility data encompass continuous numerical data—such as temperature, pressure, and vibration at the manufacturing site—and discrete status data, such as equipment status and alarms. These data are collected in real-time by PLCs and sensor

networks and transmitted via industrial protocols such as OPC-UA and MQTT [2]. Preprocessing and filtering based on edge computing improve the data quality and enable real-time monitoring with high-frequency updates within milliseconds to seconds.

MES transactional data, which relate to business processes that occur during the execution phase of manufacturing, include various types of data, such as work orders, performance, defects, material receipts and issues, and inventory. As structured relational data generated according to business processes, these data exhibit complex interrelationships. MES data must ensure the consistency of transactions, making traceability and data versioning important elements [3].

Given these characteristics of manufacturing data, traditional query-based systems face significant challenges in providing flexible and real-time access to this information. This paper proposes a novel architecture that addresses these challenges through an integrated approach combining RAG technology with domain-specific optimizations. MES plays an important role in managing information generated from various business areas such as production, quality, facilities, materials, work orders, etc. to support process optimization and faster decision-making. However, traditional MES data management approaches rely on predefined queries or template-based reports, making it difficult to respond flexibly to new queries or uncertain situations and inconvenient for non-technical users to navigate information intuitively.

Recent advances in LLM-based natural language-processing technologies provide the ability to retrieve and analyze information based on natural language queries without directly exposing complex SQL queries or data structures to the user [4]. In particular, RAG technologies do not simply rely on LLM, but dynamically reference external knowledge bases and combine them with LLM to enable more accurate and flexible response generation [5]. In the MES domain, this RAG-based approach allows for easier access to process-, machine-, and time-specific knowledge, as well as the integration of this knowledge into the SQL generation process to create a more user-friendly data utilization experience [6].

While traditional RAG solutions like LangChain provide powerful general-purpose capabilities, there are many reasons why "domain-specific" RAG components are needed for the specialized environment of MES. MES has data with unique characteristics, such as product history, real-time production data, and sensor logs, which require specialized indexing, Extract–Transform–Load (ETL) pipelines, and security policies to handle effectively [7]. There are also unique requirements for MES environments, such as real-time data processing, reflecting domain knowledge, complying with security regulations, integrating with MES systems, and utilizing domain knowledge graphs [8]. LangChain is an open source project released under the Apache License 2.0, which is free for commercial use and modification and provides stable functionality. Based on this, custom components such as MES-specific embedding models, document structure optimization, manufacturing site terminology processing, and time series data processing can be developed.

Recent advances in LLM-based natural language-processing technology have enabled natural language-based information retrieval and analysis without directly exposing complex SQL queries or data structures. In particular, RAG technology dynamically references external knowledge bases and combines with LLM to enable more accurate and flexible response generation. In the MES domain, this RAG-based approach provides a more user-friendly data utilization experience by easily accessing process, equipment, and time-specific knowledge and integrating it into the SQL generation process [9].

In this paper, we propose a practical approach for improving natural language based data accessibility in MES environments through the integration of RAG and local LLM. This study proposes a method to design a domain-specific RAG architecture optimized for

MES data processing, present efficient text-to-SQL conversion with error handling, and verify the performance of the system in real manufacturing environments.

This paper is organized into five sections. Section 2 describes the overall structure of the proposed MES-specific RAG-LLM architecture and details the roles of each component and how they interact. In Section 3, we specifically present the implementation of real-time data processing, domain knowledge integration, and error recovery mechanisms, which are the core features of the proposed architecture. In Section 4, we demonstrate the performance superiority of the proposed architecture through experimental results utilizing real-world manufacturing floor data, especially the improved query-processing accuracy, response time, and scalability compared to existing RAG solutions. Finally, Section 5 concludes this work, presents future research directions, and discusses possible further developments of the proposed architecture.

2. Materials and Methods

The core of the proposed framework consists of three main components: the data pipeline, the RAG engine, and the user interface. The data pipeline is responsible for collecting and structuring real-time and historical data from various sources such as MES and equipment sensors, while the RAG engine leverages advanced natural language-processing capabilities to interpret user queries and generate responses through a built-in search mechanism to effectively access stored data. Finally, the user interface provides an intuitive interactive experience for users to enter questions and gain real-time insights.

These components integrate seamlessly to form a data discovery framework that supports data analysis and rapid decision-making in smart manufacturing environments. Each layer of the framework is designed to effectively handle requests and responses between MES data and users throughout the entire process, from data collection to response generation. This enables fast, accurate, and informed decision-making in a smart factory environment. Figure 1 provides a unified view of the hardware infrastructure and data flow processes of the proposed MES-RAG system architecture. At the center of the system is a high-performance application server based on Ubuntu 22.04 LTS with more than 8 CPU cores and 32 GB of RAM. This server works by orchestrating three core layers.

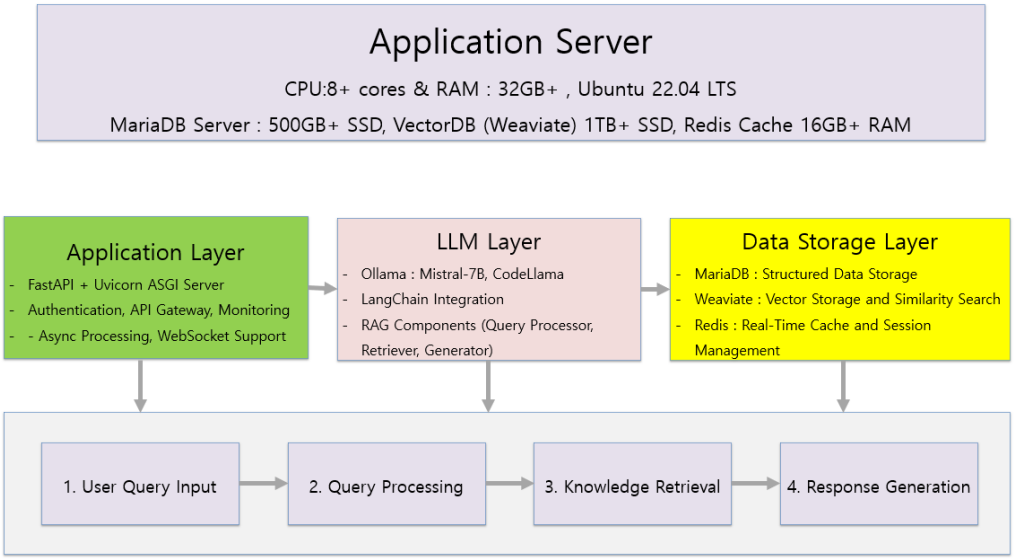


Figure 1. System overview: hardware, software, and data flow.

The application layer implements high-performance asynchronous processing through FastAPI and the Uvicorn ASGI server. This layer provides functions such as authentication,

API gateway management, and websocket support for real-time communication. The LLM layer integrates the Mistral-7B and CodeLlama models with LangChain to form a RAG pipeline. Here, the query processor analyzes and structures natural language input, the searcher performs context-based information retrieval, and the generator generates responses based on the retrieved knowledge.

There are three data storage tiers. MariaDB manages structured MES data using 500+ GB SSDs, the Weaviate vector database stores vectorized knowledge with 1+ TB SSDs, and the Redis cache utilizes 16+ GB of RAM to support real-time data access. The data flow in the system begins with user queries being entered through the application layer. The input query is processed at the LLM layer, relevant knowledge is retrieved from the storage layer, and finally a response is generated and delivered to the user. This architecture ensures efficient query processing, scalable data management, and rapid user interaction while maintaining the consistency of the data and system reliability.

2.1. Architecture Overview

The proposed framework is implemented using the hardware and software architecture shown in Figure 1, and is designed to support the increasing demand for facility data and MES data integration and statistical retrieval generated by smart factory production processes. It also applies dense path search techniques to improve retrieval accuracy, and introduces the latest Few-Shot learning method to improve model performance [10].

The hardware architecture is centered around an application server with 8+ CPU cores and 32+ GB of RAM, coupled with a MariaDB server with 500+ GB of SSD, a Weaviate vector database with 1TB of SSD, and a Redis cache server with 16+ GB of RAM. This configuration is optimized for processing large amounts of MES data in real-time and building a highly scalable infrastructure while maintaining high performance.

The software architecture consists of four main layers. First, the Application Layer is built on top of FastAPI and the Uvicorn ASGI server and provides functions such as authentication and API gateways, monitoring, asynchronous request processing, and WebSocket support. This allows users to query and analyze data in real-time through a web interface. Second, the LLM Layer uses the Ollama framework, including the Mistral-7B and CodeLlama models. This layer includes RAG components consisting of Query Processor, Retriever, and Generator, which are responsible for processing natural language queries and performing SQL transformations through integration with LangChain. Third, the Data Storage Layer is responsible for relational data storage using MariaDB, vector storage and similarity search using Weaviate, and real-time cache and session management using Redis. This hierarchy enables simultaneous structured and unstructured searches of the data and provides fast query responses [11].

Finally, the Development Environment is based on Anaconda Python 3.10 and includes key packages such as LangChain v03, Transformers 4.25.1, SQLAlchemy 2.0, and FastAPI 0.45.0 to facilitate system-wide development and maintenance. This layered structure optimizes the entire process from data collection to response generation, effectively handling real-time MES data requests and responses for users. This enables fast and accurate decision-making in a smart factory environment.

Figure 2 provides a visual representation of the overall architecture of the proposed MES-RAG system. The architecture consists of three main layers: a data source layer, a data pipeline and processing layer, and an application layer.

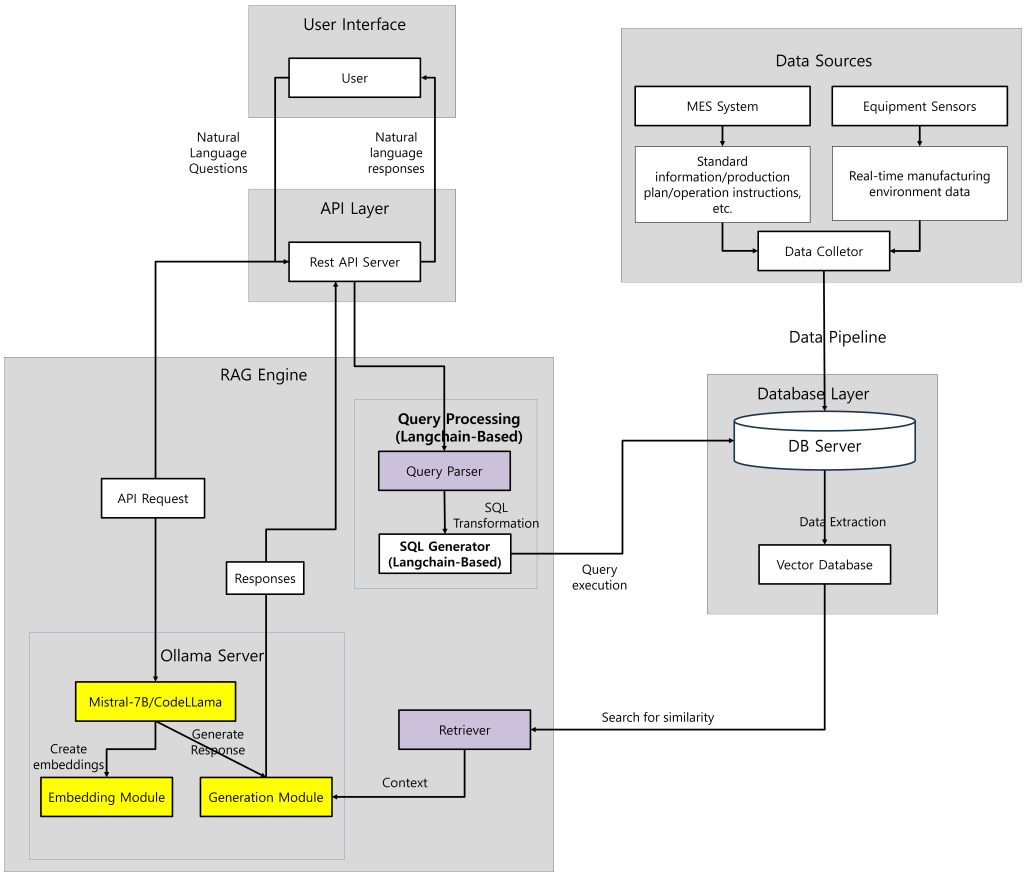


Figure 2. Key components and data flow of the MES-RAG system.

First, the Data Sources Layer collects data generated by the MES system and machine sensors. This layer collects various structured data, such as historical production records, quality information, equipment status, work orders, etc., as well as real-time manufacturing process data to ensure the continuity and consistency of the data. The collected data are processed through the Data Pipeline. During this process, the Data Collector cleanses the data and performs Vector Embedding and relational data storage to transform the data into a searchable form. This provides a structure that can support both structured (SQL-based) and unstructured (vector search) data retrieval.

The RAG Framework handles natural language-based queries. The user enters a question in natural language via the User Interface, which is analyzed by the Query Processor to retrieve the appropriate data, retrieve relevant documents, and generate a final response by the Generator. The final response is returned to the user via the Output Handler, where the LLM Layer performs natural language understanding and generation utilizing a local large language model based on Ollama. Finally, at the Application Layer, a FastAPI-based API server manages natural language query requests and coordinates interactions with the RAG engine and database. This layer is designed to optimize real-time data analytics with asynchronous request processing, API gateways, and websocket support, and to provide fast responses. This layered structure enables real-time data processing and secure data management, enabling fast and flexible data discovery and analysis in manufacturing environments.

2.2. Real-Time Data Processing

2.2.1. Classification of Manufacturing Data

Manufacturing data can be broadly classified as facility or MES data. Facility data encompass continuous numerical data—such as temperature, pressure, and vibration at

the manufacturing site—and discrete status data, such as equipment status and alarms. These data are collected in real-time by PLCs and sensor networks and transmitted via industrial protocols such as OPC-UA and MQTT. Pre-processing and filtering based on edge computing improve the data quality and enable real-time monitoring with high-frequency updates within milliseconds to seconds. Furthermore, data pre-processing is important in terms of handling noise and outliers.

MES transactional data, which relate to business processes that occur during the execution phase of manufacturing, include various types of data, such as work orders, performance, defects, material receipts and issues, and inventory. As structured relational data generated according to business processes, these data exhibit complex interrelationships. MES data must ensure the consistency of transactions, making traceability and data versioning important elements.

2.2.2. Data Processing

The data-processing architecture proposed in this study was designed as a multilayered structure to achieve both real-time performance and high-quality data [12]. First, the data collection layer employs edge computing to collect real-time data and apply various protocol conversions and normalization techniques to enhance data quality through initial filtering and pre-processing [13]. This minimizes data-processing delays, reduces the network load, and supports real-time decision making [14].

The collected data are processed in real-time through FastAPI's asynchronous handlers and Apache Kafka in the streaming data-processing layer. Apache Flink is used to validate the data streams and manage ephemeral storage, thereby ensuring the availability and consistency of the data. Redis is also used as an in-memory cache to accelerate the processing of frequently accessed data [15].

Finally, the processed data are stored in a relational database based on the MS SQL Server. This relational database is synchronized with a vector database to support the real-time search and analysis of the data. Advanced analyses—such as similarity search, anomaly detection, and clustering—can be performed using the vector database, and data retention is ensured through the backup and archiving of the data.

2.2.3. End-to-End Data Flow

The proposed architecture extracts data from the MES and production equipment; verifies data formats to ensure the accuracy of the data; ensures the integrity of data through transaction management, concurrency control, and version control to ensure the data remain consistent; and tracks the history of changes in the data. To ensure the transparency and reliability of the data, the architecture supports change-history management, log recording, and the traceability of the data to identify the creation path and clarify the responsibilities. Finally, reliability is enhanced by quantitatively assessing and improving the data's quality using various metrics that measure the completeness, accuracy, consistency, and timeliness of the data.

Figure 3 illustrates the data flow within the system. Data are ingested at the edge, processed through a streaming pipeline, and finally stored in the relational and vector databases. The figure specifically illustrates real-time data filtering at the edge computing layer, streamed processing through Apache Kafka, and performance optimization through Redis caching. This data flow ensures high performance, real-time processing, and reliable data storage.

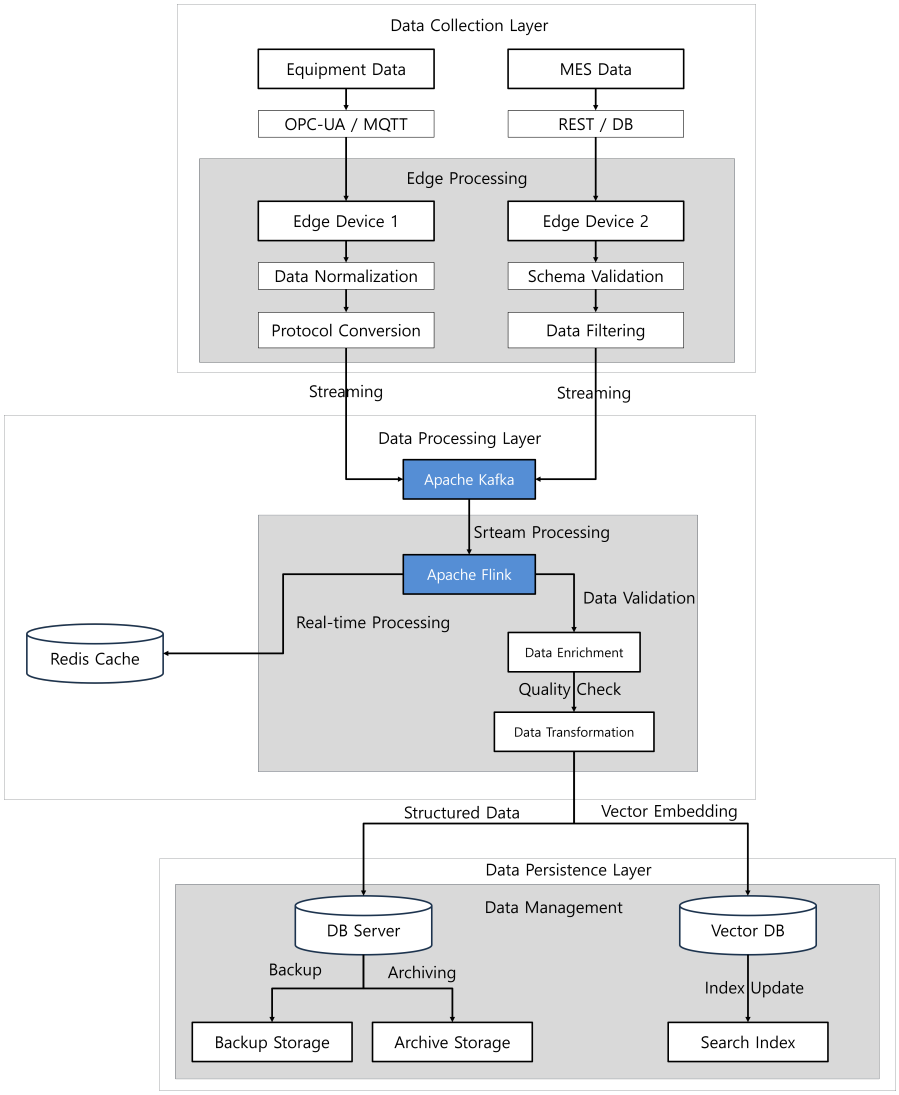


Figure 3. End-to-end data flow diagram.

2.2.4. Data Quality Assurance

The proposed architecture uses several techniques to ensure the quality of the data. First, schema, data type, and constraint validations are performed to ensure the accuracy of the data, thereby maintaining structural consistency and preventing unexpected errors. To ensure data consistency, we implemented transaction management, concurrency control, and versioning. Transactions ensure data integrity by bundling groups of operations into single logical units, concurrency control prevents data conflicts caused by simultaneous access by multiple users, and version control tracks the history of data changes, as well as enabling reversions to previous versions if necessary.

To ensure the transparency and trustworthiness of data, the architecture supports change-history management, audit log recording, and data lineage tracking. Change history management tracks changes in the data, and audit logs similarly record the changes to ensure accountability. The data lineage helps us to understand how data are created and assess their reliability. Finally, we used a variety of metrics to quantitatively assess the data quality. Data are continuously monitored and improved by measuring missing required field rates, missing value rates, data type match rates, value range compliance rates, duplicate data rates, referential integrity compliance rates, data latency, and update cycle compliance rates. This enhances the reliability of the data and improves the accuracy of data-driven decisions. At its core, the architecture ensures data quality as follows:

- Accuracy: schema, data type, and constraint validation;
- Consistency: transaction management, concurrency checking, versioning;
- Transparency: change history management, audit logs, data lineage tracking;
- Reliability: measurement of various quality metrics, continuous improvement.

2.3. RAG-Based Knowledge Retrieval

A key element of the proposed architecture is the RAG-based MES data search engine. The engine is designed to provide users with statistical and analytical data to support on-site decision-making by integrating MES data, as well as various data generated by production equipment, in real-time. It consists of three main components: a query processor, a retrieval, and a generator. Each component works complementarily to provide accurate and meaningful responses to user queries.

The query processor is responsible for converting the natural language query entered by a given user into a form that the system can understand. It maps the user's query into a vector space that the computer can process, helping it retrieve relevant information from the database [16]. The retriever searches the database for relevant information based on vectors generated by the query processor. It uses vector similarity and hybrid search to find the documents most similar to the query, and employs dynamic context windowing to provide the most contextual results. Finally, the generator takes the information retrieved by the retriever and uses the LLM to generate answers in natural, coherent sentences [17]. The generated answers represent accurate and unambiguous responses to the user's query, providing additional information if required.

The main advantage of this system is real-time responsiveness. The system retrieves relevant information and generates an answer as soon as the user enters a query. It also uses an LLM to ensure high-quality answers, and combines vector similarity and hybrid searches to maintain high relevance. In addition, it can answer different types of queries, making it flexible to user needs. The query processor converts natural-language queries entered by users into a form that the system can interpret. In other words, it maps each user's query into a vector space that the computer can process, thereby helping retrieve relevant information from the database.

- Vectorization: This converts the user's query into a high-dimensional vector that numerically represents the meaning of the query, enabling the system to measure the similarity between the query and data;
- Semantic analysis: Determines the core meaning of the query and provides accurate search results;
- Context expansion: Accounts for the query's context to add relevant information, enhancing the accuracy of the search results;
- SQL transformation: Converts natural-language queries into SQL queries for direct access to structured data;
- Hybrid query optimization: Combines structured (SQL) and unstructured (vector) data searches to improve search efficiency.

The retriever is responsible for retrieving relevant information from the database based on the vectors generated by the query processor.

- Vector similarity search: Calculates the similarities between the query vector and document vectors stored in the database to find the most relevant documents;
- Hybrid search: Combines a keyword-based search with a semantic search to increase the search's accuracy and scope;
- Dynamic context window customization: Adjusts the scope of the search based on the context of the query to deliver the most contextual results;
- Caching mechanism: Caches frequently searched query results to improve response time;

- Progressive search: Uses a phased search strategy to balance search result quality and response time.

The generator is responsible for generating responses for the user based on the information retrieved by the retriever.

- Prompt generation: Generates prompts that can be interpreted by the LLM based on the retrieved documents;
- Text generation: Deploys the LLM to generate responses consisting of natural, coherent sentences;
- Answer evaluation: Ensures quality by evaluating whether the generated answers are appropriate for the user's query, grammatically correct, etc.;
- Fact check: Verifies that the generated answers match te information in the retrieved documents;
- Source tracking: Provides the source of the original data upon which the generated answer is based.

Figure 4 shows the interactions between the main components of the RAG system, illustrating how the query processor, retriever, and generator components work together to process a user's natural language query. The query processor vectorizes the user's input, the retriever retrieves relevant information, and the generator generates the final response. Notably, the data flow and processing between each component are represented sequentially, providing a clear understanding of how the RAG system operates as a whole to enable accurate and reliable response generation.

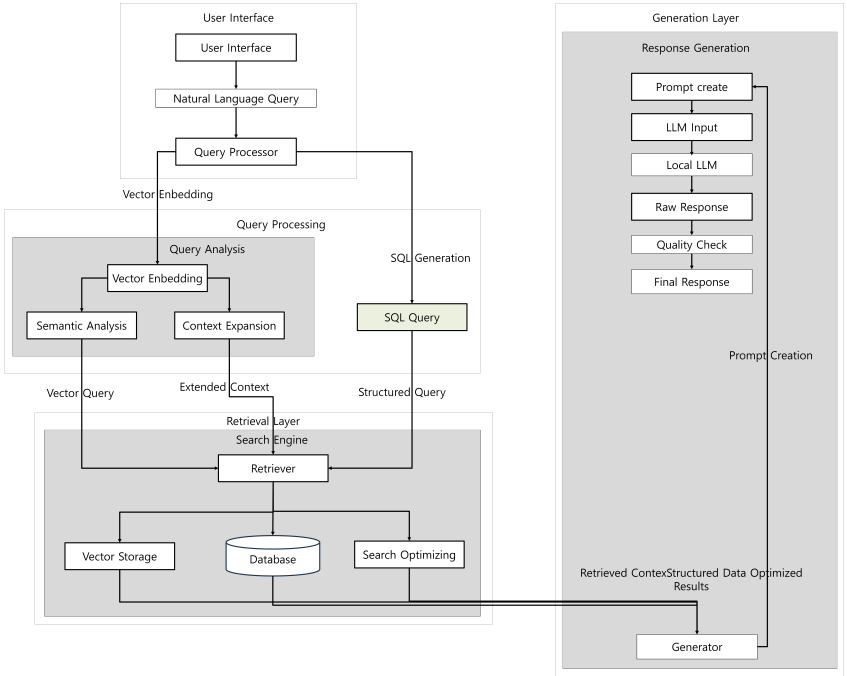


Figure 4. Interacting with RAG Components.

2.4. SQL Generation Pipeline

The SQL generation pipeline uses a multistep approach to improve accuracy. First, it determines the intent of a given natural language query and identifies the relevant tables. It then generates the necessary join conditions, constructs the WHERE clause, and finally assembles and optimizes the final query. A library of query templates specific to the MES domain was constructed to encompass frequently used query patterns such as production performance queries, equipment condition monitoring, and quality data analysis. Thus, an appropriate template is selected and used depending on the context.

2.5. User Interaction

The proposed system implements a conversational interface model that is fundamentally different from traditional MES user interfaces. While traditional MES environments typically use rigid web-based interfaces with predefined menus and fixed data query paths, our approach adopts a more flexible, conversational-based interaction pattern that improves data accessibility and user experience. The system leverages a chatbot-like interface to enable natural language communication between the user and the system. This approach allows users to express their data needs in natural language without the constraints of predefined query templates or fixed menu structures. The interface will be able to incorporate a number of key mechanisms to ensure effective communication, such as continuously maintaining conversational context to help users refine queries and explore data relationships more intuitively [18].

A typical interaction flow begins with the user submitting a natural language query through the chat interface. The system processes this query using the RAG component and may ask for clarification if necessary. Once it understands the intent of the query, it generates and presents a response in an appropriate format. The user can then follow up with related queries or request corrections to the information presented, maintaining an ongoing conversation that enables a more in-depth exploration of the data. This conversational approach offers a number of advantages over traditional interfaces. Users do not need to navigate complex menu hierarchies or understand specific query syntax; the system can guide users through query refinement when necessary and suggest relevant information based on the context of the conversation. In addition, the interface adapts to different user expertise levels, making it accessible to both technical and non-technical users while maintaining the sophisticated data-handling capabilities required in a manufacturing environment.

Error-handling and feedback mechanisms are integrated throughout the interaction process. The system provides immediate feedback on unclear or ambiguous queries and engages users in interactive error resolution through natural dialog. This approach not only helps users formulate more effective queries, but also contributes to the system's continuous learning and improvement process.

2.6. LLM Agents and RAG for Question Answering

Recently, hybrid approaches that combine LLM and RAG have gained traction in the programming domain. Lewis et al. demonstrated that such hybrid approaches outperform traditional LLMs by 15–25% on knowledge-intensive tasks [6]. In particular, there is active research on improving the quality and accuracy of programming questions, which is attributed to the effective combination of code generation and document retrieval. The key to hybrid RAG systems is the balance between accurate information retrieval and context-based response generation. Barron et al. proposed an architecture that combines a domain-specific vector store with a knowledge graph [11]. This approach leverages multi-layered information sources such as code snippets, API documentation, and technical specifications to achieve 83.5% recall. Dynamic search scope adjustment based on question intent and domain-specific embeddings for precision search are key factors in this achievement.

The context quality metrics proposed by Wang et al. play an important role in the pre-verification of code quality and executability [12]. The memory efficiency optimization achieved through dynamic context windowing significantly improved the scalability of the system, and the in-context learning and self-correction mechanism developed by Pourreza and Rafiei improved code generation accuracy by 15–20% [19]. In terms of relevance to this research, the PDF document-processing optimization techniques proposed by Yang et al. directly influenced the design of our system's document-processing pipeline [20].

Notably, the multi-column layout recognition and table information extraction accuracy improvement methodologies have become core components of our system.

Given the specificity of the programming domain, domain knowledge integration is crucial to the design of RAG systems. Developing specialized vector embeddings for each programming language and effectively linking API documentation and code repositories are key. Real-time versioning and updating to reflect the latest information is also essential. In terms of context optimization, dynamic composition with token efficiency is key. The process of reconfiguring the context based on the intent of the question and integrating different forms of information determines the performance of the system. The quality metrics proposed by Iftikhar et al. are an important criterion to ensure the reliability and executability of the code generated in this process [19].

We evaluated and optimized the performance of our system based on the domain-specific benchmarks proposed by Wang et al. and further improved the retrieval accuracy by 18% by adopting the tensor decomposition-based search method proposed by Barron et al. An automated code review system, execution result-based validation, and integration of user feedback enable continuous quality improvements.

3. Results

In this paper presents the necessity and efficiency of RAG- and LLM-based text-to-SQL transformation in the MES environment of smart factories. The proposed architecture can be implemented through Docker and Anaconda virtualization environments and reflects the data-processing requirements of real manufacturing sites. Particular emphasis was placed on the accuracy of domain-specific query processing, the reliability of error recovery mechanisms, and the real-time response performance.

3.1. Related Works

In this paper, we analyze the results of existing research to explore the performance improvement potential of introducing an RAG system and text-to-SQL technology in a MES environment.

RAG techniques combine the ability to generate LLM with external knowledge discovery to generate more accurate and contextualized responses. The paper “Optimizing RAG Techniques for Automotive Industry PDF Chatbots: A Case Study with Locally Deployed Ollama Models” found that, compared to simple RAG models, the advanced RAG system with a customized context compression pipeline significantly improved performance [17]. Specifically, the introduction of self-supervised RAG agents improved answer relevance by 7.3%, 6.3%, and 9.4%, and increased fidelity by 4.6%, 6.8%, and 5.5% on QReCC, CoQA, and self-built datasets, respectively. Contextual accuracy increased by 2.2%, 3.3%, and 2.4%, and contextual recall increased by 2.4%, 2.1%, and 3.3% [17].

In addition, the study “RAG VS FINE-TUNING: Knowledge Injection into Pretrained Language Models” analyzed the combined effect of RAG and fine-tuning. The accuracy of the baseline model improved from 47% to 72% with fine-tuning and increased to 74% when RAG was added [21]. This shows that the combination of fine-tuning and RAG can have a synergistic effect.

The study “Optimizing RAG Techniques for Automotive Industry PDF Chatbots: A Case Study with Locally Deployed Ollama Models” found that introducing a custom function call mechanism to optimize the output of Ollama models resulted in an additional 9.0–13.3% improvement in answer relevance, 6.0–7.1% improvement in fidelity, 1.9–5.3% improvement in contextual accuracy, and 4.6–5.6% improvement in contextual recall [17].

To improve the accuracy of text-to-SQL conversion, recent studies have tried various approaches, including Intermediate Representations (IRNet), Structure-Based Dictionary

Learning (STRUG), and the application of LLM. The experimental results of these studies support the design direction of the system we propose in this work. First, the experiments with IRNet showed an overall accuracy improvement of 19.5% compared to the existing SyntaxSQLNet, and a significant performance improvement of 23.3% when the difficult level was Hard, which requires complex SQL structures [22]. This demonstrates that intermediate representations are effective in improving SQL transformation accuracy in a variety of domains. A study applying the STRUG technique achieved 5.5% higher accuracy than the default RAT-SQL model when evaluating the Spider development set. In particular, it showed a performance improvement of 8.4% when the difficultly level was Extra Hard, and the largest improvement in WHERE clause processing from 71.7% to 75.6% [23]. It also maintained 5.7% higher execution accuracy than the BERT-based model on the realistic Spider-Realistic dataset [23]. A recent study utilizing a large language model showed even more remarkable results. The CodeLLaMA-34B model achieved an average accuracy of 20.6% higher than the existing Text-to-SQL model, and outperformed LLaMA-2-CHAT-70B with twice the number of parameters by 12.4% [24]. In particular, one-shot evaluations using GPT-4 achieved up to 80.7% accuracy, depending on the prompt design [24]. These prior studies show the effectiveness of combining intermediate representations with large language models to improve the accuracy of text-to-SQL conversions. The system we propose in this study was also designed based on this approach, and further performance improvements are expected to be achieved by combining it with RAG.

3.2. Experiment Setup

To implement the proposed architecture, we planed a virtual experiment environment based on Docker container and Anaconda for reproducibility and scalability. The MariaDB database and Weaviate vector database were run as containers, and a Python library environment was built in a virtual environment. This setup can allowed to run experiments and verify results under the same conditions, regardless of the hardware specifications or host OS.

The Docker container-based environment was set up to simultaneously run a relational DB and establish a Weaviate environment for the vector search. MariaDB created an initial DB by specifying DB_ROOT_PASSWORD and DB_NAME as environment variables, and loaded tables and sample data for testing via init.sql. Weaviate utilized the latest images to provide a vector embedding-based search (or RAG), and we set the search limits by specifying QUERY_DEFAULTS_LIMIT as an environment variable. By organizing it as a container, multiple services can be managed in isolation without depending on specific libraries, allowing for easy extension or updates in the future.

We created a virtual environment with Anaconda to install major libraries such as FastAPI, Uvicorn, SQLAlchemy, and Weaviate-client, which is important for the following reasons. First, it minimizes unpredictable errors or fluctuations in results by maintaining the compatibility of the versions of the libraries used, regardless of various OS environments. Second, it ensures reproducibility so that other researchers or practitioners can perform experiments under the same conditions by configuring the environment with the same version.

Libraries such as Transformers and LangChain are key components of the local LLM and RAG, used to automate text-to-SQL operations and compute contextual information. By disclosing the above configuration, we can clearly show how the core elements of the proposed architecture (LLM, vector DB, relational DB) interwork, and simplify the installation and configuration procedures for further research or industrial applications. Furthermore, by providing precise version information and execution methods, reproducibility is ensured, allowing for other researchers to achieve the same results and providing a practical guide for implementation or extension.

Finally, the proposed structure suggests that the Mistral-7B and CodeLlama-7B models can be utilized in different roles. It is effective to configure Mistral-7B to be responsible for natural language query understanding and context processing, while CodeLlama-7B is responsible for SQL generation. Both models can be run in a local environment through the Ollama framework, and memory usage can be optimized through the application of 8-bit quantization.

Building SQL query–natural language pair datasets from real-world manufacturing sites is important for configuring models specific to MES environments. These datasets are preferably collected by major business areas, such as production management, quality management, and facility management [25]. Database schema information can be provided in JSON format and should include information about the relationships and constraints between tables. When applying the few-shot prompting technique, it is effective to utilize examples that reflect the specificity of the MES domain. For production, quality, and facility-related queries, including representative examples of each in the prompts enables domain-specific responses to be generated. For handling SQL errors, consider including common error patterns and solutions in the prompts.

3.3. Experiment Methodology

To examine the effectiveness and efficiency of the proposed system, this paper focuses on metrics such as response time, query accuracy, system overhead, precision, speed, and fault tolerance to validate the effectiveness of Text-to-SQL conversion and establish a baseline for comparison with existing MES query systems [26]. Metrics are utilized as a key tool to quantitatively evaluate system performance, identify problems, and highlight the advantages of the proposed system over existing systems. In particular, in text-to-SQL conversion systems, metrics play a role in evaluating performance, analyzing efficiency, verifying reliability, and ensuring comparability. These metrics can be expected to provide quantitative verification, suggest improvement directions, enhance user experience, support real-time decision-making, and provide a basis for comparison. To verify the conversion effectiveness of the Text-to-SQL system, a set of natural language queries for each major business area of the actual manufacturing site was organized. Table 1 shows the test scenarios and evaluation metrics for each business area.

Table 1. MES domain-specific test scenarios and evaluation metrics.

Business Area	Test Case Types	Evaluation Points	Example Queries
Production Management	Performance analysis	Multi-table join, aggregation function	“Production volume and target achievement rate of each product on each line yesterday”
	Quality analysis	Time series processing, grouping	Processes with high frequency of defects by type in the last month”
Materials Management	Inventory Analysis	Conditional filtering, threshold processing	“Unordered items of material below safety stock”
	LOT tracking	hierarchy processing, traceability	“Raw material LOT information of defective finished products”
Facilities Management	Utilization analysis	Real-time data processing	“Which facilities are currently under 70% utilization and Causes”
	Predictive maintenance	Time series pattern analysis	“List of facilities scheduled for next scheduled maintenance”

The transformation performance for each query is evaluated from three aspects: schema-matching accuracy, temporal condition handling, and domain rule reflection. Table 2 shows the detailed evaluation items for each area of evaluation.

Table 2. Text-to-SQL conversion evaluation framework.

Evaluation Area	Evaluation Items	Evaluation Method	Key Indicators
Functional Evaluation	Query-processing accuracy	domain-specific standard query set validation	Schema-matching rate, business rule reflection
	Time-processing accuracy	Time series data-processing validation	Time point interpretation accuracy, period-processing accuracy
	Domain knowledge utilization	Manufacturing-specific requirement validation	Manufacturing terminology interpretation rate, rule application accuracy
Non-functional Evaluation	System performance	System resource monitoring	Response time, CPU/memory usage
	Scalability	Load testing	Concurrent user throughput, data-processing volume
	Stability	Error recovery testing	Automatic recovery rate, system availability

For comparison with existing MES query systems, we set the evaluation criteria as shown in Table 3. This provided objective criteria to demonstrate the practical value of the system.

The proposed evaluation methodology provides a basis for comprehensive system validation during the actual implementation phase. Future work will build on this framework to perform an empirical validation in various manufacturing environments.

Table 3. Comparison analysis with existing MES systems.

Evaluation Item	Existing MES System	Proposed System	Key Differences
Response Time	Template-based fast but slows with complex queries	Consistent response time with RAG-based real-time search	Consistent performance for complex requests via Redis caching and dynamic query processing
Query Accuracy	Dependent on fixed templates, weak with new queries	Enhanced accuracy with domain knowledge reflection	Accurate reflection of domain-specific data structures and user intent
Fault Tolerance	Manual correction needed for errors	Automatic recovery mechanism provided	Enhanced stability through automatic schema-matching and time error recovery
User Accessibility	SQL knowledge required	Natural language-based queries possible	System easily usable by non-experts
System Overhead	High CPU and memory usage	Reduced resource consumption with optimized processing	Efficient resource utilization for large-scale data processing

3.4. Scalability and Performance Metrics

The proposed architecture implements both horizontal and vertical scaling mechanisms to ensure system performance under increasing data volumes and concurrent user

queries. In terms of horizontal scalability, the system leverages distributed data processing via MariaDB's primary–secondary replication configuration to enable the distributed processing of read operations [27]. The vector database component utilizes Weaviate's sharding capabilities for the distributed storage and retrieval of large vector data, and the Redis cluster provides an elastic scaling of cache capacity and throughput.

Vertical scalability is achieved through optimized resource allocation for each system component. The application server is configured with 8+ CPU cores and 32+ GB of RAM to ensure sufficient concurrent request-processing capacity. The Vector database utilizes SSD storage to ensure fast search speeds, and the in-memory cache is allocated more than 16 GB of RAM to optimize the cache hit rates. The system's concurrent processing is implemented through FastAPI's asynchronous processing capabilities that efficiently manage concurrent requests. WebSocket support supports distributed real-time data-processing loads, and connection pooling optimizes database connection management. Load-balancing is achieved through request routing and load-balancing via API gateways, and the cache tier reduces thee database load by separating read/write operations.

Data-processing optimization is implemented through several mechanisms. Vector-embedding batch processing improves system resource efficiency, while optimized indexing strategies improve query performance. Time series data partitioning improves search efficiency for temporal data queries [28]. These optimization strategies work together to maintain a consistent performance as data volumes increase. Performance monitoring and management is accomplished by comprehensively tracking key metrics, as shown in Table 4. These metrics provide a quantitative measure of system performance and scalability across multiple dimensions, including response performance, system resource utilization, database performance, and concurrent handling capacity.

Table 4. Performance and scalability monitoring metrics.

Monitoring Area	Key Metrics	Target	Monitoring Method
Response Performance	Average Response Time	Simple Query: <1 s Complex Query: <3 s	APM Tools
System Resources	CPU, Memory Usage	CPU: < 70% Memory: < 80%	System Monitoring
Database	Query Throughput Cache Hit Rate	TPS: 1000+ Cache Hit Rate: 80%+	DB Monitoring Tools
Concurrency	Concurrent Users Requests per Second	100+ Users 500+ Requests/s	Load Testing Tools

Future scalability improvements will focus on three main areas: First, a distributed processing framework will be implemented that integrates Apache Kafka for real-time data pipelines and Apache Flink for extended streaming data processing. Second, high-availability configurations will be enhanced through the deployment of multiple availability zones and the improvement of automatic recovery mechanisms. Third, improvements in performance optimization will be achieved through query pattern analysis and caching algorithms.

With these scalability design principles and performance optimization strategies, we aim to maintain reliable service delivery even as the amount of data and the number of users increase. The monitoring framework provides continuous feedback for system optimization, enabling proactive scaling decisions based on actual usage patterns and performance metrics.

3.5. Data Processing for MES-RAG Integration

3.5.1. Hierarchical Processing Structure

The proposed data-processing architecture is tiered, flexible, scalable, and can operate in real-time. It overcomes the limitations of traditional MES data management structures by taking into account the heterogeneous nature of plant and MES data while implementing optimized processing and storage strategies.

At a foundational level, edge computing-based ingestion processes multi-protocol data via OPC-UA, MQTT, and similar methods at the point closest to the production floor. It performs preprocessing tasks including data normalization, protocol conversion, and noise filtering to improve data quality and reduce transmission load. This enables the efficient handling of high-frequency updates and low-latency data pipelines to enable real-time decision making [29].

In the next step, we implement streaming data processing through a high-performance pipeline using Apache Kafka and Apache Flink to perform real-time data validation, fine-grained quality verification, and enrichment. The Redis in-memory cache improves performance and responsiveness across the data-processing layer by providing fast access to frequently requested information. This tier ensures the availability, consistency, and timeliness of the data for real-time analytics and advanced inferences such as LLM-based data searches and RAG.

The final layer integrates the data persistence and vector database. After preprocessing and streaming analytics, structured data are stored in a relational database such as MS SQL Server to support structured data analysis, transaction history management, and the reflection of business logic. These data are concurrently associated with a vector database to enable unstructured data exploration and analysis, including similarity searches, anomaly detection, and vector embedding-based clustering. A long-term retention strategy with backup, archiving, and active storage ensures reliable management and scalability throughout the data lifecycle.

This tiered processing structure organically combines real-time streaming processing, structured data management, and unstructured searches and analytics to provide a holistic response to the different types of data generated in an MES environment. Ultimately, this is the foundation for enabling natural language-based queries and high-level insights using LLM and RAG.

3.5.2. Data Pipeline

The data pipeline workflow consists of several integration steps. The process starts with extracting real-time and cyclic data from MES and equipment, followed by denoising, outlier detection, protocol conversion, and edge layer normalization. For real-time validation and quality assurance, the system uses Kafka and Flink to perform data validation, quality checks, and transformations through streaming pipelines to maximize the completeness, consistency, and accuracy of the data.

Redis provides caching and fast in-memory processing to improve responsiveness. The transaction management phase stores the cleansed data in a relational database with transaction consistency, concurrency control, versioning, and change history tracking mechanisms to ensure reliable data access for business decisions. The system also supports vector embedding and unstructured analytics by embedding data in a vector database in parallel with SQL Server, allowing users to explore data relationships through natural language queries and perform semantic analysis through LLM and RAG-based approaches.

The pipeline maintains continuous quality assessments and improvements through real-time feedback loops based on data quality metrics, facilitating continuous performance

enhancements, quality control, and model retraining to improve the overall reliability and maintainability of the data management and analytics system.

These workflows fulfill the purpose of the architecture: to build high-quality, real-time data pipelines and provide intuitive data access through LLM and RAG, which transcends the limitations of traditional SQL-based queries. The rationale for this architectural design is to establish an intelligent data management and analytics ecosystem that provides rich insights and rapid response with minimal expertise, even in MES environments.

3.6. Text-to-SQL Conversion

To implement text-to-SQL conversion, we first constructed a set of MES domain-specific queries. Our objective was to improve the accuracy of SQL conversion for natural language queries by selecting queries in key business areas (production management, materials management, and task management) that are frequently raised in manufacturing sites and performing instruction tuning based on these queries.

3.6.1. MES Query Set

The query set for each category within the MES domain was designed to include simple queries at the aggregation level as well as complex queries, such as those that identify correlations between quality factors. This enables the local LLM to perform SQL transformations that are domain-specific and not simply based on keyword-matching. Statement tuning is performed by expanding the original query to be more specific, or by inserting additional conditions that leverage domain-specific knowledge. For example, a simple query such as, "What was the production volume yesterday?" can be rewritten as a richer contextual query—such as, "How many units of each product did each production line produce yesterday and how well did they meet their targets?"—and an appropriate SQL query can be presented to guide the model in clearly understanding the target concept and generating sophisticated SQL. This instruction tuning process improves query accuracy by encouraging the local LLM to reflect conditions such as the production line, product, time period, and target versus goal in the SQL.

3.6.2. Error Analysis and Recovery

Even if a set of queries specific to the MES environment is constructed, various errors can occur in the actual operating environment because of incomplete queries or unfamiliarity with the environment. To prevent and quickly respond to these errors, we selected frequently occurring error types and applied the same treatment method as that used for the instruction tuning. The errors were categorized as schema linkage or time-processing errors, and appropriate correction patterns were presented for each error case.

This error handling pattern can also be applied to new queries in a manner similar to statement tuning. For example, given a new query such as 'average uptime per asset for the last week', existing time-processing patterns and asset-specific schema structures can be used to impose appropriate time-range filtering and join conditions. RAG and the LLM enable the automation or semi-automation of these error-handling patterns, providing a flexible basis for responding to new domain queries and error situations. Consequently, given the complex query requirements and error scenarios encountered in MES environments, the instruction-tuning and error-handling strategies proposed in this study support continuous performance improvements and enhance practical applicability.

The following are the main types of SQL errors that can occur in an MES environment and how to avoid them. Table 5 presents specific examples of SQL errors commonly encountered in a Manufacturing Execution System (MES) environment. It demonstrates how incorrect queries can lead to issues like table join errors, time-processing errors, and aggregation errors. For each error type, the table provides an "Incorrect Query", an

"Improved Query" that addresses the error, and the "Key Improvements" made to correct the query. Table 6 broadens the scope by outlining common error types in MES SQL queries, the typical issues associated with each type, and the countermeasures that can be implemented to prevent these errors. The table also details the expected positive effects of these countermeasures, such as enhanced data integrity, consistent time management, and improved query reliability.

Table 5. SQL Error examples and improvements in MES environment.

Error Type	Incorrect Query	Improved Query	Key Improvements
Table Join Error	SELECT * FROM PRODUCTION p, QUALITY q WHERE p.date = '2024-01-01'	SELECT * FROM PRODUCTION p INNER JOIN QUALITY q ON p.lot_id = q.lot_id WHERE p.date = '2024-01-01'	Explicit join condition using metadata-driven relationship mapping
Time Processing Error	WHERE create_time = GETDATE()	WHERE CAST(create_time AS DATE) = CAST(GETDATE() AS DATE)	Standardized date format handling with proper type-casting
Aggregation Error	SELECT line_id, SUM(quantity), product_name FROM PRODUCTION	SELECT line_id, SUM(quantity), product_name FROM PRODUCTION GROUP BY line_id, product_name	Proper GROUP BY clause for aggregate functions with non-aggregated columns

In SELECT *, * means "all columns".

Table 6. Error types and countermeasures in MES environment.

Error Type	Common Issues	Countermeasures	Expected Effects
Table Join Errors	Missing join conditions Incorrect relationship mapping	Metadata-based relationship management Automated join condition generation	Enhanced data integrity Reduced join-related failures
Time Processing Errors	Timezone inconsistency Date format mismatches	Standardized time handling library Temporal pattern formalization	Consistent time management Improved temporal queries
Aggregation Function Errors	Invalid GROUP BY clauses Missing aggregation fields	Automated GROUP BY validation Required field verification	Accurate statistical analysis Reliable aggregation results
Syntax Errors	SQL syntax violations Query structure issues	Automated syntax validation Error pattern database	Enhanced query reliability Systematic error prevention

3.6.3. Query Processing Pipeline Validation

In this section, we illustrate our query-processing pipeline using a combination of RAG and LLM methods. The proposed architecture dynamically references MES domain knowledge through the RAG during the process of converting natural language queries to SQL, and allows the LLM to use this context to form the correct SQL patterns. The process encompasses two phases: context retrieval and SQL pattern-matching.

We validate the effectiveness of the proposed architecture in response to novel, non-predefined types of queries. Our objective was to verify that novel queries can be handled by reusing existing instruction tuning patterns and error handling strategies, and that the proposed architecture adapts to complex requirements with different conditions and time scales.

These validation cases demonstrate that the proposed RAG-based LLM can adaptively handle various requirements beyond simple queries, such as complex conditions, temporal patterns, and ratio calculations. Instruction-tuning strategies and error-handling patterns augmented with domain-specific knowledge can be consistently applied to new queries, achieving a high level of query interpretation and transformation ability that effectively reflects the complex data structures and business logic of MES environments.

4. Discussion

In this study, we developed a text-to-SQL transformation using RAG and a local LLM in an MES environment, and identified important implications based on the experimental results obtained from the proposed architecture. First, we clearly demonstrated that the RAG system can effectively reflect the specificity of the MES domain in the SQL generation process. When processing queries specific to each business domain—such as production management, quality control, and facility management—the use of domain-specific knowledge contributes significantly to the improvement of SQL conversion accuracy. This suggests that it is possible to reflect the data structures and business logic of specific industries beyond the domain-adaptation limitations of existing generic text-to-SQL conversions.

We also found that the use of an Ollama-based local LLM practically addresses concerns related to data privacy and system responsiveness [30]. The ability to generate complex SQL queries without exposing sensitive manufacturing site data is an important factor that enhances the proposed architecture's applicability. Furthermore, the error recovery mechanism proposed in this study was shown to be effective in systematically identifying and resolving common errors in MES environments, including schema linkage and time-processing errors. This is significant because it provides a practical method to deal with various types of errors that may occur during SQL generation. However, this study had several limitations. The experimental data did not sufficiently cover all the business areas of real-world MES, and there were limitations in reflecting the specificities of different industries. In addition, a validation in large-scale concurrent user environments or long-term operational situations was not sufficiently performed, and the inability to use large-scale models, stemming from computational resource constraints, limits the potential to validate model performance changes.

Despite these limitations, this study is significant in that it provides a concrete and useful solution to the practical challenge of improving natural language data accessibility in MES environments. Future research should explore various optimization directions, including the standardization of performance evaluation methods, validation for different industrial domains, the application of larger-scale models and distributed processing architectures, and the improvement of caching mechanisms. The practicality and scalability of the system may be further improved by extending the system to different manufacturing industries, supporting multilingualism, and integrating multiple databases.

To further illustrate the advances made by our proposed framework, Table 7 presents a comprehensive comparison between traditional approaches and our solution in MES environments. This comparison reveals several key advantages of our framework. First, the implementation of domain-specific RAG-based natural language processing significantly enhances query flexibility while maintaining high accuracy. Second, our architecture's distributed processing capabilities and containerization ensure both real-time performance

and scalability. Third, the automatic error recovery mechanism and local model deployment address critical industrial requirements for reliability and data security.

The comparison demonstrates that while AI-enhanced systems have made progress in user accessibility, they often lack domain-specific optimizations and may compromise on data security due to external AI dependencies. Our framework addresses these limitations while maintaining the benefits of advanced natural language processing. Particularly noteworthy is the framework's ability to combine high security standards typical of traditional MES with the user-friendly interface of modern AI systems, all while adding domain-specific optimizations for manufacturing environments.

Table 7. Comparison of Data Analysis Approaches in MES Environment.

Characteristics	Traditional MES	Template-Based Systems	AI-Enhanced Systems	Proposed Framework
Query Method	Fixed SQL queries	Predefined templates	Basic natural language processing	Domain-specific RAG-based NLP
Real-time Processing	Limited	Partial support	Supported	Fully supported with distributed processing
Scalability	Low	Medium	Medium	High (containerization, distributed processing)
Error Handling	Manual processing	Template-based processing	Basic automation	Automated recovery mechanism
Data Security	High	High	Medium (external AI dependency)	High (local model usage)
Domain Knowledge	Limited	Medium	Limited	High (MES-specific vector search)
User Accessibility	Low (expert required)	Medium	High	High (natural language interface)
Maintainability	Low	Medium	Medium	High (modular design)

5. Conclusions

We developed a text-to-SQL conversion architecture for MES environments using RAG and a local LLM to enable intuitive data access based on natural language while reflecting the domain specificity of manufacturing sites. We found that complex manufacturing data structures can be accurately transformed into the SQL by implementing a query-processing specific to each MES business domain and leveraging the RAG to effectively utilize the domain knowledge. Furthermore, we systematically classified the SQL generation error types commonly encountered in MES environments, such as schema-matching and timing errors, and implemented a recovery mechanism that can actively respond to various error situations. We confirmed that the Ollama-based local LLM can be utilized to prevent the leakage of sensitive production-site data, addressing a critical issue in real-world industrial applications.

The effectiveness of the proposed architecture was verified in terms of domain-specific query-processing accuracy, the reliability of the error recovery mechanism, real-time response performance, and scalability. However, challenges such as long-term validation in real industrial environments, performance verification based on large datasets, and scalability in different manufacturing domains, remain. In future studies, it will be necessary to establish a standardized benchmark dataset to establish a performance evaluation methodology, develop objective performance evaluation metrics, and conduct verifications

in different industries. In addition, further approaches to system optimization are possible, such as applying large-scale language models, implementing distributed processing architectures, and improving caching mechanisms.

Future work will extend this research in three main directions. First, in terms of integrating advanced AI techniques, we will enhance the decision support system on the manufacturing floor by leveraging the inference capabilities of large-scale language models. We will optimize the local LLM based on Ollama for the manufacturing domain to improve its performance and extend it to allow for it to handle non-textual data through multimodal analysis. We will also enhance the analytical capabilities of the system by adding advanced analytical functions such as time series prediction and anomaly detection.

Second, we will validate and improve the industry-specific scalability of the proposed architecture. We will evolve the system to reflect the characteristics of different manufacturing industries, such as food, electronics, and automotive industries. This includes effectively integrating the unique production processes, quality control requirements, regulatory compliance, etc., of each industry into the system. In particular, we will study how to systematically structure and utilize industry-specific domain knowledge in a vector database.

Third, we plan to improve the error recovery mechanism to increase the stability and reliability of the system. Beyond the current schema-matching and time-processing error recovery, we will develop an intelligent error-handling system that automatically detects and responds to various anomalies on the production floor. To this end, we will introduce a deep learning-based anomaly detection model and improve the system by learning possible error patterns to which it could proactively respond.

Author Contributions: Conceptualization, H.C. and J.J.; methodology, H.C.; architecture, H.C.; validation, H.C. and J.J.; formal analysis, H.C.; research, H.C.; acquisition of data, H.C.; data curation, H.C.; drafting of the manuscript, H.C.; revision and editing of the manuscript, H.C. and J.J.; visualization, H.C.; supervision, J.J.; project management, J.J. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding. The article publication charge (APC) was waived as part of the publisher's special discount programme.

Data Availability Statement: This research proposes a theoretical architectural framework and no data has been separately generated or analysed. The sample queries and Structured Query Language (SQL) statements presented in this paper are descriptive examples to illustrate the functionality of the proposed architecture. The referenced database schema and table structure are based on a typical MES implementation, but are not related to any specific real-world dataset. The architecture design and implementation details are described in the main body of the thesis.

Acknowledgments: This work was supported by ICT Creative Consilience Program through the Institute of Information & Communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (RS-2020-II201821).

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Bokrantz, J.; Skoogh, A.; Lämkull, D.; Atieh, H.; Perera, T. Data Quality Problems in Discrete Event Simulation of Manufacturing Operations. *Simul. Trans. Soc. Model. Simul. Int.* **2018**, *94*, 1009–1025. [CrossRef]
2. Chen, B.; Wan, J.; Shu, L.; Li, P.; Mukherjee, M.; Yin, B. Smart Factory of Industry 4.0: Key Technologies, Application Case, and Challenges. *IEEE Access* **2017**, *6*, 6505–6519. [CrossRef]
3. Shojaeinasab, A.; Charter, T.; Jalayer, M.; Khadivi, M.; Ogunfowora, O.; Raiyani, N.; Yaghoubi, M.; Najjaran, H. Intelligent Manufacturing Execution Systems: A Systematic Review. *J. Manuf. Syst.* **2022**, *62*, 503–522. [CrossRef]

4. Ning, Z.; Tian, Y.; Zhang, Z.; Zhang, T.; Li, T.J.-J. Insights into Natural Language Database Query Errors. *ACM Trans. Interact. Intell. Syst.* **2024**, *14*, 25. [CrossRef]
5. Kanburoğlu, A.B.; Tek, F.B. Text-to-SQL: A Methodical Review of Challenges and Models. *Turk. J. Electr. Eng. Comput. Sci.* **2024**, *32*, 403–419. [CrossRef]
6. Lewis, P.; Perez, E.; Piktus, A.; Petroni, F.; Karpukhin, V.; Goyal, N.; Küttler, H.; Lewis, M.; Yih, W.-T.; Rocktäschel, T.; et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In Proceedings of the 34th Conference on Neural Information Processing Systems (NeurIPS 2020), Vancouver, BC, Canada, 6–12 December 2020; pp. 945–964.
7. Chan, A. GPT-3 and InstructGPT: Technological Dystopianism, Utopianism, and "Contextual" Perspectives in AI Ethics and Industry. *AI Ethics* **2023**, *3*, 53–64. [CrossRef]
8. Abro, A.A.; Talpur, M.S.H.; Jumani, A.K. Natural Language Processing Challenges and Issues: A Literature Review. *Gazi Univ. J. Sci.* **2023**, *36*, 1522–1536. [CrossRef]
9. Fürst, J.; Kosten, C.; Nooralahzadeh, F.; Zhang, Y.; Stockinger, K. Evaluating the Data Model Robustness of Text-to-SQL Systems Based on Real User Queries. *ACM Trans. Database Syst.* **2024**, *49*, 1–24.
10. Brown, T.B.; Mann, B.; Ryder, N.; Subbiah, M.; Kaplan, J. Language Models are Few-Shot Learners. *arXiv* **2020**, arXiv: 2005.14165.
11. Barron, R.C.; Grantcharov, V.; Wanna, S.; Eren, M.E.; Bhattarai, M.; Solovyev, N.; Tompkins, G.; Nicholas, C.; Rasmussen, K.Ø.; Matuszek, C.; et al. Domain-Specific Retrieval-Augmented Generation Using Vector Stores, Knowledge Graphs, and Tensor Factorization. *arXiv* **2024**, arXiv:2410.02721.
12. Wang, S.; Liu, J.; Song, S.; Cheng, J.; Fu, Y.; Guo, P.; Fang, K.; Zhu, Y.; Dou, Z. DomainRAG: A Chinese Benchmark for Evaluating Domain-Specific Retrieval-Augmented Generation. *Chin. Nat. Lang. Process.* **2023** , 2024, 89–98.
13. Ali, M.I.; Patel, P.; Breslin, J.G. Middleware for Real-Time Event Detection and Predictive Analytics in Smart Manufacturing. In Proceedings of the 2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS), Santorini, Greece, 29–31 May 2019; pp. 370–376.
14. Al-Gumaei, K.; Müller, A.; Weskamp, J.N.; Longo, C.S.; Pethig, F.; Windmann, S. Scalable Analytics Platform for Machine Learning in Smart Production Systems. In Proceedings of the 2019 24th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), Zaragoza, Spain, 10–13 September 2021; pp. 1155–1162.
15. Li, X.; Wan, J.; Dai, H.-N.; Imran, M.; Xia, M.; Celesti, A. A Hybrid Computing Solution and Resource Scheduling Strategy for Edge Computing in Smart Manufacturing. *IEEE Trans. Ind. Inform.* **2019**, *15*, 4225–4234. [CrossRef]
16. Pan, J.J.; Wang, J.; Li, G. Vector Database Management Techniques and Systems. In Proceedings of the Companion of the 2024 International Conference on Management of Data, Santiago, Chile, 9–15 June 2024; pp. 597–604.
17. Liu, F.; Kang, Z.; Han, X. Optimizing RAG Techniques for Automotive Industry PDF Chatbots. In Proceedings of the 2024 3rd International Conference on Artificial Intelligence and Intelligent Information Processing, Tianjin China, 25–27 October 2024; pp. 152–159.
18. Singh, A.; Shetty, A.; Ehtesham, A.; Kumar, S.; Khoei, T.T. A Survey of Large Language Model-Based Generative AI for Text-to-SQL-Benchmarks, Applications, Use Cases, and Challenges. *arXiv* **2024**, arXiv: 2412.0528.
19. Pourreza, M.; Rafiei, D. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. *arXiv* **2023**, arXiv:2304.11015.
20. Zhang, L.; Liu, Y.; Fan, Z.; Dai, H. Empower Large Language Model to Perform Better on Industrial Domain-Specific Question Answering. *IEEE Trans. Ind. Inform.* **2024**, *20*, 1523–1538.
21. Wang, B.; Chen, H.; Xu, T.; Li, M.; Zhou, J. RAG vs. Fine-tuning: Pipelines, Tradeoffs, and a Case Study on Agriculture. *Comput. Electron. Agric.* **2024**, *213*, 108156.
22. Deng, X.; Awadallah, A. H.; Meek, C.; Polozov, O.; Sun, H.; Richardson, M. Structure-grounded pretraining for text-to-SQL. *arXiv* **2020**, arXiv:2010.12773.
23. Gao, D.; Wang, H.; Li, Y.; Sun, X.; Qian, Y.; Ding, B.; Zhou, J. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. *arXiv* **2023**, arXiv: 2308.15363. [CrossRef]
24. Guo, J.; Zhan, Z.; Gao, Y.; Xiao, Y.; Lou, J.G.; Liu, T.; Zhang, D. Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation. In Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics, Florence, Italy, 28 July–2 August 2019; ACL: Stroudsburg, PA, USA, 2019; pp. 4524–4535.
25. Zhao, W.X.; Zhou, K.; Li, J.; Tang, T.; Wang, X.; Hou, Y.; Min, Y.; Zhang, B.; Zhang, J.; Dong, Z.; et al. A Survey of Large Language Models. *arXiv* **2024**, arXiv: 2303.18223.
26. Iftikhar, N.; Baattrup-Andersen, T.; Nordbjerg, F.E.; Bobolea, E.; Radu, P.-B. Data Analytics for Smart Manufacturing: A Case Study. In Proceedings of the 8th International Conference on Data Science, Technology and Applications, Prague, Czech Republic, 26–28 July 2019; pp. 392–399.
27. Henzel, R.; Herzwurm, G. Cloud Manufacturing: A State-of-the-art Survey of Current Issues. *Procedia CIRP* **2018**, *72*, 947–952. [CrossRef]
28. Jing, Z.; Su, Y.; Han, Y. When Large Language Models Meet Vector Databases: A Survey. *arXiv* **2024**, arXiv: 2402.01763.

29. Lee, J.; Bagheri, B.; Kao, H.-A. A Cyber-Physical Systems Architecture for Industry 4.0-Based Manufacturing Systems. *Manuf. Lett.* **2015**, *3*, 18–23. [CrossRef]
30. Chen, L.; Zhang, W.; Smith, K. LLM-Based Edge Intelligence: A Comprehensive Survey on Architectures, Applications, Security and Trustworthiness. *IEEE Commun. Surv. Tutor.* **2024**, *26*, 1–28.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.