Institute for Machine Tools and Industrial Management (*iwb*)
TUM School of Engineering and Design
Technical University of Munich

Development of a Large Language Model for Interfacing Tool Condition Monitoring Data

Entwicklung eines Large-Language-Modells für den Zugriff auf Werkzeugzustandsüberwachungs-Daten

Scientific work for obtaining the academic degree

Master of Science (M.Sc.)

at the TUM School of Engineering and Design of the Technical University of Munich

Examined by Prof. Dr.-Ing. Michael Zäh
Chair of Machine Tools and Manufacturing Technology
Submitted by Ana Carolina Ramos Cunha
Munich, Germany
Submitted on March 31, 2026 in Garching

Institute for Machine Tools and Industrial Management (*iwb*)
TUM School of Engineering and Design
Technical University of Munich

Task Description

English Title of the Bachelor's/Semester/Master's Thesis:

Development of a Large Language Model for Interfacing Tool Condition Monitoring Data

German Title of the Bachelor's/Semester/Master's Thesis:

Entwicklung eines Large-Language-Modells für den Zugriff auf Werkzeugzustandsüberw
Daten

Written by:	B. Sc. Ana Carolina Ramos Cunha
Matriculation-No.:	03811403
Degree Program:	M. Sc. Informatics
TUM School:	TUM School of Computation, Information and Technology
Advised by:	M. Sc. Daniel Piendl
Time period:	01.10.2025 to 31.03.2026

Initial Situation
In digitalized production environments, large quantities of data are continuously generated by machine tools, sensors, and monitoring systems. At the Institute for Machine Tools and Industrial Management, detailed data from tool condition monitoring experiments are collected and stored in a database. These data contain valuable information on tool wear, process parameters, and in-process sensor measurements.

However, the current access to these data typically requires expert knowledge of database structures and query languages. As a result, extracting relevant insights from the database is time-consuming and limited to technically skilled users.

With the rapid progress in artificial intelligence, particularly in large language models, there is growing potential to bridge this gap by enabling intuitive natural-language interaction with industrial databases. Such a system would make production data directly accessible to engineers, researchers, and operators, thereby facilitating data-driven decision-making and knowledge extraction in production environments.

Objective
The objective of this interdisciplinary project is to design and implement an LLM-based question-answering system capable of retrieving and interpreting information from the tool condition monitoring database. The system shall allow users to input natural-language questions and receive precise, data-backed answers derived from the underlying database. Ultimately, the project aims to demonstrate how AI-driven systems can improve the accessibility and usability of complex production data and support intelligent production decision-making.

Approach and Working Methodology

- **System design:** Define the system architecture and the data flow between the milling tool database, the LLM, and the user interface.
- **Data preparation:** Preprocess the database contents to create a structured dataset for analysis.
- **LLM integration:** Implement a retrieval-augmented generation pipeline connecting the processed data with an LLM.
- **Interface development:** Create a simple web-based interface for user interaction.
- **Evaluation:** Test the system using representative production queries and improve the system based on the observed performance.

Agreement
With the supervision of B. Sc. Ana Carolina Ramos Cunha by M. Sc. Daniel Piendl, the intellectual property of the *iwb* is incorporated into this work. Publication of the work or disclosure to third parties requires the permission of the chair holder. I agree to the archiving of the thesis in the *iwb*'s library, which is only accessible to *iwb* employees, and in the *iwb*'s digital thesis database as a PDF document.

Garching, 30.09.2025

Prof. Dr.-Ing. Michael Zäh

B. Sc. Ana Carolina Ramos Cunha

Abstract

This project develops and evaluates a large-language-model-based question-answering system for tool condition monitoring data stored in a PostgreSQL database. The main objective is to enable natural-language access to production and process data without requiring users to write SQL queries or understand database internals.

The implemented solution combines a FastAPI backend, a LangGraph-based workflow, Redis-backed conversation memory, and a Next.js web interface. User requests are classified, mapped to relevant database tables, translated into SQL, validated for safety, executed in read-only mode, and summarized in natural language.

The resulting prototype demonstrates that large language models can provide a practical interface for engineers and researchers to retrieve data-backed insights from industrial databases. In addition, a benchmark and test framework was implemented to assess robustness, latency, and answer behavior across different model configurations. The project shows that LLM-assisted database interaction can improve the accessibility of industrial data while maintaining important operational safeguards.

Contents

List of Figures

List of Tables

Chapter 1 Introduction

Modern manufacturing environments generate large volumes of tool condition monitoring data during production. These data are valuable for process understanding, quality assurance, and operational decision support. However, effective access to such data is often limited to users who possess detailed knowledge of database schemas and query languages.

At the Institute for Machine Tools and Industrial Management, this creates a practical gap between available information and efficient day-to-day usage. Engineers and researchers may know which insights they need, but obtaining them can require time-consuming manual interaction with the underlying database. This project addresses that problem by providing a natural-language interface that allows users to ask questions directly and receive structured, data-backed answers.

1.1 Initial Situation and Motivation

In modern production environments, large amounts of data are continuously collected from machine tools, sensors, and monitoring systems. These data contain relevant information on tool wear, process behavior, and sensor measurements and can therefore support data-driven engineering decisions. Nevertheless, access to such information is often restricted by the need for SQL knowledge and familiarity with the underlying database schema.

The motivation of this project is to reduce that barrier and provide a more intuitive way of interacting with industrial data. A natural-language interface based on large language models offers the possibility of enabling engineers, researchers, and operators to retrieve insights without directly formulating database queries. The implemented project addresses this need through a retrieval-augmented generation pipeline built on a PostgreSQL database and a modern web-based software stack (Next.js Contributors, 2025; PostgreSQL Global Development Group, 2025; Ramirez, 2025).

1.2 Objective and Scope

The objective of this interdisciplinary project is to design and implement an LLM-based question-answering system for tool condition monitoring data. The system shall:

- accept natural-language questions about tool condition monitoring data,
- identify relevant database structures,
- retrieve the required information through automatically generated SQL queries,
- return concise and understandable answers, and

- support iterative interactions by considering conversation context.

The scope of the project includes system design, schema preparation, backend and frontend implementation, workflow orchestration, and systematic evaluation. The project does not aim to replace direct expert database work in all cases, but rather to provide a practical and safe interface for common information requests.

1.2.1 Research Contribution

The main contributions of this work are an end-to-end natural-language-to-SQL assistant architecture, schema-grounded prompt generation, safety validation for SQL execution, an interactive web-based prototype, and a benchmark setup for evaluating model behavior and system robustness. The workflow orchestration and model integration rely in particular on LangGraph, LangChain-related components, and Ollama-based model access (LangChain, Inc., 2025a, 2025b; Ollama, 2025).

1.2.2 Report Structure

This report is structured as follows. Chapter 2 presents the methodology, system architecture, and implementation details. Chapter 3 discusses evaluation, limitations, conclusions, and future work. Additional documentation on AI-based tool usage and supplementary notes are provided in the appendix.

1.2.3 Project Context

The project is situated at the intersection of artificial intelligence, database interaction, and industrial production engineering. It demonstrates how language models can support knowledge extraction from production data while preserving operational safeguards through controlled SQL generation and execution.

1.3 Summary of Chapter

The introduction outlined the initial situation, the motivation for natural-language database access, and the objective of the project. It also summarized the main contribution and positioned the work within the broader context of industrial AI applications.

Chapter 2 Methodology and System Design

This chapter presents the methodological foundation of the project, the system architecture, and the core implementation decisions. The final solution is implemented as a retrieval-augmented generation (RAG) system that enables natural-language interaction with tool condition monitoring data.

2.1 Background on LLM-Based Database Question Answering

Large language models can be used to transform natural-language requests into executable SQL queries. In industrial environments, however, such systems must satisfy requirements that go beyond simple text generation. They must be grounded in the actual database schema, restricted to safe read-only access, and robust against ambiguity, incomplete requests, and hallucinated database references.

The project objective was to implement a retrieval-augmented generation pipeline connecting processed database information with a large language model. In the implemented system, retrieval is realized primarily through schema-aware grounding, table relevance selection, relationship expansion, and Redis-based conversation memory. Consequently, the final implementation can be described as a RAG pipeline in which retrieval is based on structured database schema information, relevant table selection, and conversational context. The implementation draws on the capabilities of PostgreSQL for structured data access, Redis for state management, and LangGraph for workflow orchestration (LangChain, Inc., 2025b; PostgreSQL Global Development Group, 2025; Redis Ltd., 2025).

2.2 Overall Architecture

The implemented system consists of several coordinated components:

- **Frontend:** A Next.js-based chat interface for user interaction, conversation management, and model-mode selection (Next.js Contributors, 2025).
- **Backend:** A FastAPI application that provides API endpoints, request validation, and workflow orchestration (Ramirez, 2025).
- **Workflow engine:** A LangGraph-based processing pipeline that decomposes the question-answering task into modular stages (LangChain, Inc., 2025b).
- **Persistence layer:** PostgreSQL for tool condition monitoring data and Redis for conversation state and memory (PostgreSQL Global Development Group, 2025; Redis Ltd., 2025).

- **Model integration:** Support for local and cloud-capable LLM execution through Ollama-based model access (Ollama, 2025).

2.3 Processing Pipeline

For each user query, the system executes the following sequence of steps:

1. classify the request as a database-related question or a general chat query,
2. merge relevant information from recent conversation history,
3. select relevant tables based on schema-aware relevance scoring,
4. generate an SQL query constrained by the selected schema context,
5. validate the generated query for safety and plausibility,
6. execute the query in a read-only PostgreSQL transaction,
7. retry with bounded correction attempts if recoverable SQL errors occur, and
8. formulate the final answer in natural language.

This structure allows the system to separate reasoning, validation, and execution clearly and thereby improve transparency and maintainability. The overall processing pipeline is illustrated in Figure 2.1.

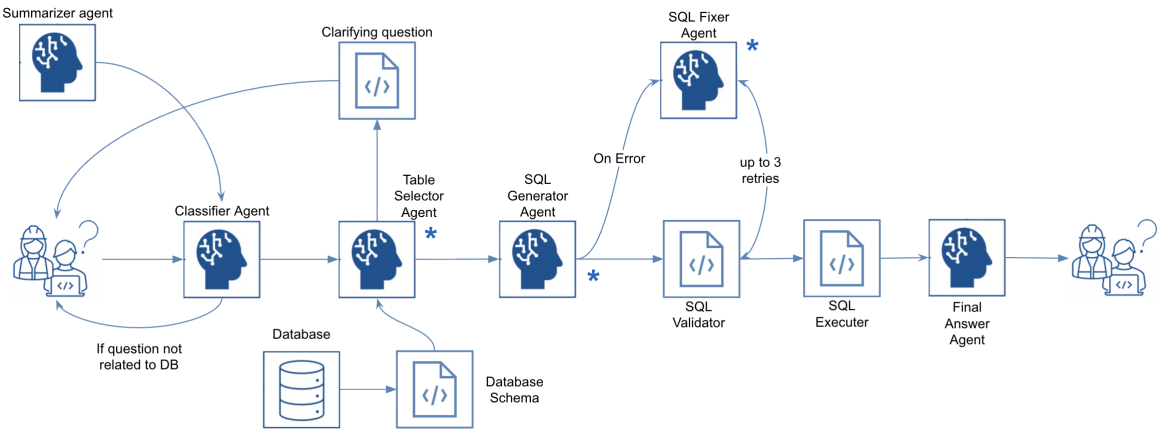

Figure 2.1: Workflow of the RAG-based question-answering pipeline

2.3.1 Schema Preparation and Grounding

Reliable SQL generation depends strongly on accurate schema information. For that reason, the project includes a schema extraction process that generates machine-readable and human-readable schema representations. These contain available tables, columns, and foreign-key relations.

The extracted metadata are used to limit the prompt scope, provide join hints, and reduce hallucinated references to non-existent database elements. This schema grounding is one of the key design choices for improving robustness and forms an important part of the retrieval stage of the RAG pipeline.

2.3.2 Safety and Robustness Measures

The system includes multiple layers of protection:

- validation of each SQL statement before execution,
- rejection of non-read-only statements and suspicious patterns,
- allowlisting of schema elements based on the selected context,
- database-side execution in read-only mode with timeouts, and
- bounded retry logic for recoverable generation errors.

These safeguards are essential in industrial settings because users must be able to trust that the system does not perform unsafe operations and that answers remain grounded in the available data.

2.3.3 User Interface Design

The frontend provides a chat-based interaction model that makes database access more intuitive for non-specialist users. In addition to the question-answering interface, it supports conversation creation, switching, renaming, and deletion. It also includes practical actions such as copying, reloading, and exporting responses.

A model-mode switch enables experimentation between local and cloud-backed inference modes without changing the basic user experience. This supports both development and deployment use cases.

2.4 Implementation

The backend was implemented using FastAPI and exposes endpoints for health checks, chat requests, and conversation management. It handles authentication checks, client tracking, conversation identifiers, and communication with the workflow engine (Ramirez, 2025).

The workflow itself is implemented as a set of modular nodes, each responsible for a specific transformation step such as classification, context handling, SQL generation, validation, execution, or answer rendering. This modularity simplifies testing, debugging, and future extension of the system (LangChain, Inc., 2025a, 2025b).

On the frontend side, the Next.js application integrates the chat interface with backend routes and manages conversation synchronization and error propagation. Additional development support is provided through containerized deployment, command-line tooling, and environment-variable-based configuration (Next.js Contributors, 2025).

2.5 Summary of Chapter

This chapter described the methodological and technical foundations of the implemented system. The final architecture combines a RAG-based processing pipeline with a chat-based user interface and several safety mechanisms to support robust and practical access to industrial monitoring data.

Chapter 3 Evaluation, Discussion, and Conclusion

This chapter summarizes the evaluation strategy, discusses the achieved results, identifies limitations, and outlines possible future improvements.

3.1 Evaluation Strategy

The system was evaluated on three levels:

- **Unit and integration tests:** verification of API behavior, workflow routing, and node logic,
- **Ground-truth prompt runs:** execution of representative natural-language queries with exported results for inspection,
- **Benchmark runs:** comparative experiments across model configurations using standardized logging.

This combination of tests supports both functional verification and practical assessment of answer quality, latency, and system behavior.

3.2 Evaluation Metrics

The benchmark and testing setup collects several metrics, including:

- answer mode distributions such as data-backed answer, insufficient data, or error,
- end-to-end latency, including mean and percentile-based measures,
- indicators for SQL generation and database execution errors,
- row counts and categorical output labels, and
- generated SQL statements and natural-language answers for manual review.

3.2.1 Results

The implemented prototype demonstrates that natural-language access to tool condition monitoring data is feasible in practice. Users can formulate database-related questions without manually

writing SQL, while the system automatically performs classification, schema grounding, SQL generation, validation, execution, and answer summarization.

In addition, the benchmark and test framework provides a structured basis for comparing model configurations and identifying potential weaknesses. Table 3.1 summarizes the benchmark results presented in the final project presentation.

Table 3.1: Benchmark results across different model configurations

Model	Avg. Latency (s)	Errors (%)	Accuracy (%)
gemma3:4b	> 180	81.25	12.5
llama3.2:latest	> 180	81.25	12.5
qwen3.5:2b	> 180	87.5	6.25
gpt-oss:20b-cloud	45	0	87.5

The benchmark results show a clear difference between local and cloud-based model configurations. The local models exhibited very high latency, exceeding 180 seconds on average, and showed poor accuracy combined with high error rates. In contrast, the cloud-based `gpt-oss:20b-cloud` configuration achieved the best overall performance, with an average latency of 45 seconds, no observed errors, and an accuracy of 87.5%. These results indicate that model quality has a strong influence on the reliability of the complete RAG pipeline.

3.2.2 Achievement of Objectives

The central project objective was achieved. The final system enables users to retrieve information from an industrial PostgreSQL database through natural language and receive data-backed answers without direct SQL authoring. The work therefore demonstrates the practical applicability of LLM-assisted access to production data and fulfills the interdisciplinary goal of combining AI methods, data engineering, and user-oriented software development.

3.2.3 Mapping to Original Milestones

Table 3.2 compares the milestones defined in the original task description with the implementation outcome.

3.3 Discussion

The project demonstrates that large language models can significantly improve the accessibility of industrial data when they are combined with controlled workflow orchestration and schema-aware grounding. At the same time, the implementation highlights that unrestricted natural-language-to-SQL generation would not be sufficiently reliable for an industrial setting without additional safeguards.

The design choices made in this project show that robustness depends strongly on explicit schema metadata, modular processing stages, and strict validation before query execution. The balance between usability and control is therefore a defining aspect of such systems.

Table 3.2: Mapping of original project milestones to implementation status

Original milestone	Implementation status
Project scope and architecture finalized	Completed
Training dataset prepared	Implemented as schema extraction and curated database metadata for prompting
Vector database and retrieval pipeline established	Implemented as a retrieval pipeline using schema-aware table selection and contextual grounding within the RAG framework
RAG pipeline and baseline LLM integration completed	Completed as a retrieval-augmented generation pipeline for natural-language interaction with the database
Model evaluated	Completed through tests, representative query runs, and benchmark scripts
Interactive prototype completed	Completed with a Next.js chat interface and conversation management
Documentation and user testing completed	Documentation completed; formal user testing evidence can be added if available
Final project results presented	Completed in the final project presentation

3.3.1 Limitations

Despite the successful prototype, several limitations remain:

- limited use of vector-based retrieval compared to structured schema-based retrieval,
- some evaluation outputs still require manual qualitative review,
- answer quality and latency depend strongly on the selected model configuration,
- schema updates require regeneration or refresh of grounding assets.

3.3.2 Lessons Learned

Several important lessons emerged from the project:

- strict validation and read-only execution constraints are essential for reliable natural-language database interaction,
- schema quality and metadata completeness strongly influence answer quality,
- modular workflow nodes improve debuggability and maintainability,
- automated benchmark support is necessary for model-selection decisions.

3.4 Conclusion and Outlook

This interdisciplinary project delivers a full-stack LLM-assisted interface for tool condition monitoring data and demonstrates its applicability through testing and benchmarking support. The system improves accessibility of industrial data while preserving operational safeguards through controlled SQL generation and read-only execution.

Future work should focus on extending the system beyond the current implementation. Promising next steps include expanding vector-based retrieval for mixed structured and unstructured knowledge, adding semantic answer-quality scoring, introducing finer-grained access-control mechanisms, and extending domain adaptation through richer industrial terminology and ontology support.

Bibliography

LangChain, Inc., (2025a). Langchain documentation [Official documentation]. https://python.langchain.com/docs/introduction/

LangChain, Inc., (2025b). Langgraph documentation [Official documentation]. https://langchain-ai.github.io/langgraph/

Next.js Contributors, (2025). Next.js documentation [Official documentation]. https://nextjs.org/docs

Ollama, (2025). Ollama documentation [Official documentation]. https://ollama.com/

PostgreSQL Global Development Group, (2025). Postgresql documentation [Official documentation]. https://www.postgresql.org/docs/

Ramirez, S., (2025). Fastapi documentation [Official documentation]. https://fastapi.tiangolo.com/

Redis Ltd., (2025). Redis documentation [Official documentation]. https://redis.io/docs/

Declaration of the Use of AI-based Tools

AI-based Tool	Last date of access	Use Case	Scope	Remarks
ChatGPT	March 2026	Text revision, formulation support, and partial code-related assistance	Selected implementation tasks and selected report sections	All generated content was reviewed, adapted, and validated by the author before inclusion.

Appendix A Supplementary Material

This appendix summarizes additional aspects of the project that support interpretation of the final implementation and its relation to the original task description.

A.1 Original Schedule

The original project schedule defined the following work packages:

- define the project scope, architecture, and requirements,
- prepare and clean the dataset for embeddings,
- generate embeddings and set up a vector database,
- build an RAG pipeline and integrate the LLM,
- evaluate and refine the model performance,
- develop an interactive prototype and deploy the system,
- conduct user testing and documentation,
- reserve buffer time for debugging, fine-tuning, and testing, and
- prepare the final presentation and handover.

Declaration of Independent Work

I hereby declare that this thesis is entirely the result of my own work except where otherwise indicated. I have only used the resources given in the list of references.

Garching, March 31, 2026

(Signature)