

Master's Programme in Mechanical Engineering

Impact of Ontologies and Data Structures on Retrieval Augmented Generation Sys- tems in Manufacturing Simulation Soft- ware

Milan Gautam

**Master's thesis
2025**

Copyright ©2025 Milan Gautam

Author Milan Gautam		
Title of thesis Impact of Ontologies and Data Structures on Retrieval Augmented Generation Systems in Manufacturing Simulation Software		
Programme Mechanical Engineering		
Major Mechatronics		
Thesis supervisor Professor of Practice Jenni Pippuri-Mäkeläinen		
Thesis advisor(s) Mika Anttila, MSc Tech. and Yang Chao, MSc Tech.		
Collaborative partner Visual Components		
Date 24.11.2025	Number of pages 94	Language English

Abstract

Manufacturing simulation software enables engineers to model, analyse, and optimise production systems using both structured and unstructured data. These platforms support virtual representations of manufacturing processes, equipment, and workflows, relying on extensive libraries of simulation components to capture domain specific knowledge. In such environments, effective data management and retrieval are essential for informed decision making.

This thesis investigates how ontologies and data structures affect the performance of Large Language Models (LLMs) in data from manufacturing simulation software, focusing on Retrieval-Augmented Generation (RAG) systems. To investigate, three case studies were conducted. The first evaluated RAG systems using FAISS vector stores, comparing local and cloud based embedding models on proprietary eCatalog data, and showed that schema rich, structured data and advanced prompt engineering significantly improved retrieval accuracy, with cloud-based models outperforming local alternatives, especially for complex, implicit queries. The second explored knowledge graph-based retrieval using Neo4j, assessing various retriever architectures, including hybrid approaches that combine vector, full-text, and graph traversal methods. It was found that knowledge graphs paired with hybrid retrieval strategies excel at handling context rich and relational queries. The third investigated LLM-powered agents interacting with a relational SQLite3 database enhanced with full text and vector indexes, where integrating semantic search into traditional SQL querying substantially improved performance on semantically complex queries, with more advanced agents delivering superior results at the cost of increased engineering complexity.

Across all case studies, the experiments consistently revealed that the interplay between data structure, embedding model quality, and retrieval architecture is critical to the success of RAG systems in manufacturing simulation. The research concludes that investing in robust data modelling, leveraging ontologies and knowledge graphs, and adopting hybrid retrieval strategies are essential for building effective, context-aware RAG solutions in this domain. These insights provide practical guidance for researchers and practitioners seeking to deploy RAG applications in complex engineering environments.

Keywords LLMs, RAG, Ontologies, Knowledge Graphs, Data Structures, Manufacturing Simulation

Table of contents

Preface and acknowledgements	7
Symbols and abbreviations.....	8
Symbols	8
Abbreviations	8
List of Figures	10
List of Tables	12
1 Introduction	13
1.1 Background	13
1.2 Research problem	15
1.3 Objectives	15
1.4 Scope	16
1.5 Thesis outline	16
2 Literature review	17
2.1 Manufacturing simulation software	17
2.2 Data, information and knowledge in engineering.....	17
2.2.1 Knowledge types in engineering simulation and their user correlation	20
2.3 Information retrieval and semantic technologies	21
2.3.1 Classical IR models.....	21
2.3.2 Semantic technologies in information retrieval.....	22
2.4 Evolution of artificial intelligence	22
2.4.1 Technology stack for generative artificial intelligence	24
2.4.2 LLMs	25
2.4.3 Pre-training process of LLMs	26
2.5 RAG foundations.....	29
2.5.1 RAG system architecture and core components	29
2.5.2 Data types and management in RAG	31
2.5.3 Query taxonomy in RAG systems.....	33
2.5.4 Query complexity framework and RAG techniques.....	33
2.5.5 Static versus dynamic RAG applications	34

2.5.6	Orchestration frameworks for LLM service implementation	35
2.5.7	Prompt engineering for RAG systems.....	35
2.6	Applied research on LLMs, RAG, and ontologies in manufacturing simulation.....	36
2.6.1	LLMs and RAG in manufacturing simulation.....	37
2.6.2	The role of ontologies in manufacturing simulation	37
3	Research material and methods.....	40
3.1	Research design	40
3.2	Data sources and preparation.....	41
3.2.1	QA dataset generation pipeline	42
3.3	Common technological stack used	43
3.4	Experimentation frameworks.....	45
3.4.1	FAISS vector store-based RAG.....	45
3.4.2	GraphRAG with Neo4j.....	54
3.4.3	Agentic RAG with relational database(SQLite)	60
3.5	Evaluation metrics and criteria	64
4	Results	66
4.1	Case study 1: FAISS vector store-based RAG	66
4.1.1	FAISS vector store-based RAG (JSON)	66
4.1.2	Vector retrieval with RDF data.....	71
4.2	Case study 2: GraphRAG with Neo4j.....	72
4.2.1	Result of Text2CypherRetriever.....	72
4.2.2	Results from the hybrid retriever case	73
4.2.3	Result of hybrid cypher retriever	73
4.2.4	Result of vector retriever.....	74
4.2.5	Overview on Graph RAG with Neo4j	74
4.3	Case study 3: Agentic RAG with relational database(SQLite)	75
4.3.1	Result of simple SQL agent	75
4.3.2	Result of enhanced SQL agent	75
4.3.3	Result of vector enhanced SQL agent	76
4.3.4	Overview of agentic interaction with relational database.....	76
4.4	Conclusion from the case studies	77

4.4.1	Case study 1: FAISS vector store-based RAG (Local and cloud models)	77
4.4.2	Case study 2: GraphRAG with Neo4j	78
4.4.3	Case study 3: Agentic RAG with SQLite3 database	78
5	Discussion and conclusions	79
5.1	Interpretation of results and summary of findings	79
5.2	Practical implications for manufacturing simulation	80
5.3	Answers to research questions	82
5.4	Academic and practical contributions	84
5.5	Limitations of the study and future research	84
5.5.1	Limitations of the study	84
5.5.2	Directions for future research	85
	AI Tools used during thesis	87
	References	88

Preface and acknowledgements

I am deeply grateful to Professor of Practice Jenni Pippuri-Mäkeläinen, whose thoughtful guidance and encouragement have been a constant source of inspiration throughout my thesis journey. My sincere thanks to my advisors, Mika Anttila, MSc Tech., and Yang Chao, MSc Tech., for their generous advice and practical insights, which have helped me navigate many challenges along the way.

I would especially like to thank my wife Binam for her unwavering support, patience, and understanding, which have sustained me through both the demanding and rewarding moments of this work. My family's belief in me has given me the strength to persevere, and I am truly thankful for their encouragement.

Finally, I wish to acknowledge my colleagues in Visual Components, whose collaboration, insightful conversations, and everyday kindness have enriched my research experience and made this achievement possible. To all who have supported me, I offer my heartfelt thanks.

This thesis has been partly funded through the European Commission's Horizon Europe Innovation Action programs CONVERGING (Grant Agreement 101058521) and RE4DY (Grant Agreement 101058384).

Otaniemi, 24th November 2025
Milan Gautam

Symbols and abbreviations

Symbols

Symbol Meaning/Definition

A	Generated answer (in RAG system formalism)
D	External data source (in RAG system formalism)
f	LLM application mapping function in RAG
Q	User query (in RAG system formalism)

Abbreviations

Abbreviation Full Term / Meaning

AGV	Automated Guided Vehicle
ANN	Approximate Nearest Neighbor
API	Application Programming Interface
BERT	Bidirectional Encoder Representations from Transformers
BFO	Basic Formal Ontology
BM25	Best Matching 25 (retrieval function)
CAD	Computer-Aided Design
CAE	Computer Aided Engineering
CAM	Computer-Aided Manufacturing
CAPEX	Capital Expenditure
CORA	Core Ontology for Robotics and Automation
CSV	Comma-Separated Values
DES	Discrete Event Simulation
DeMO	Discrete Manufacturing Ontology
DIKW	Data-Information-Knowledge-Wisdom (hierarchy)
DPO	Direct Preference Optimization
DTS	Digital Twin Systems
ERP	Enterprise Resource Planning
ETL	Extract, Transform, Load
ExtruOnt	Extruder Ontology
F1	F1 Score (harmonic mean of precision and recall)
FoF	Factory of the Future Ontology
FTS	Full Text Search
FTS5	Full Text Search version 5 (SQLite extension)
GPU	Graphics Processing Unit
GPT	Generative Pre-trained Transformer
HNSW	Hierarchical Navigable Small World (graph)
IOF	Industrial Ontologies Foundry
IPU	Intelligence Processing Unit
JSON	JavaScript Object Notation
LLaMA	Large Language Model Meta AI
LLM	Large Language Model
LoRA	Low-Rank Adaptation
MASON	Manufacturing's Semantics Ontology
MES	Manufacturing Execution System
MMR	Maximal Marginal Relevance
MPMO	Manufacturing Predictive Maintenance Ontology

MRO	Manufacturing Reference Ontology
NER	Named Entity Recognition
NPU	Neural Processing Unit
OEE	Overall Equipment Effectiveness
OLP	Offline Programming of Robots
PEFT	Parameter-Efficient Fine-Tuning
QA	Question-Answer (dataset)
RAG	Retrieval-Augmented Generation
RDF	Resource Description Framework
RLHF	Reinforcement Learning from Human Feedback
SOHO	Sharework Ontology for Human-Robot Collaboration
SQL	Structured Query Language
SQL Server	Microsoft SQL Server
T5	Text-to-Text Transfer Transformer
TF-IDF	Term Frequency-Inverse Document Frequency
TPU	Tensor Processing Unit

List of Figures

Figure 1 How does a simulation software engineer gain skill in particular software?	13
Figure 2 Initial concept of smart assistant.	14
Figure 3 Overview on LLM models.	14
Figure 4 How data transforms into knowledge?	18
Figure 5 Application of DIKW(Rowley, 2007) hierarchy in 3D manufacturing simulation.	19
Figure 6 How ML/AI can abstract information extraction and knowledge creation?	20
Figure 7 Knowledge types and user progression in simulation software systems	21
Figure 8 Evolution of Artificial Intelligence (Bommasani et al., 2022).	23
Figure 9 Generative AI technology stack.	25
Figure 10 LLM pre-training process pipeline.	27
Figure 11 Generic RAG architecture (S. Zhao et al., 2024).	30
Figure 12 Example of embedding representation of text.	32
Figure 13 RAG query complexity and retrieval techniques framework (based on S. Zhao et al., 2024).	34
Figure 14. Types of ontologies and examples for manufacturing and factory simulation domain (Sapel et al., 2024).	38
Figure 15 Overview of three case studies generic flow.	40
Figure 16 Implicit and explicit QA dataset generator.	42
Figure 17 Sample and structure of QA pairs.	43
Figure 18 Illustrates the transformation of the raw data corpus into the hybrid vector store.	46
Figure 19 Query to evaluation pipeline.	47
Figure 20 Data loading implementation python code snippet.	48
Figure 21 Hybrid retrieval implementation python code snippet.	48
Figure 22 Ensemble retrieval process python code snippet.	49
Figure 23 Document accumulation with token limits python code snippet.	49
Figure 24 LLM invocation python code snippet.	49
Figure 25 Post processing loop python code snippet.	50
Figure 26 Evaluation metrics python code snippet.	50
Figure 27 Result aggregation python code snippet.	50
Figure 28 Immediate human feedback implementation.	51
Figure 29 Sample of final excel workbook with multiple sheets.	51
Figure 30 JSON to RDF data ETL pipeline.	52
Figure 31 Overall architecture of FAISS vector store-based RAG starting data corpus as RDF serialized file.	53
Figure 32 Structured flat document format.	53
Figure 33 Snippet from flat turtle document structure.	54
Figure 34 Schema based prompt template for both structured and turtle format documents embedded.	54
Figure 35 Program flow of creating Neo4j database out of SQLite database.	55
Figure 36 Schema of the database.	55

Figure 37 Ecatalog metadata as knowledge graph in Neo4j.....	56
Figure 38 Vector index creation workflow for Neo4j database.....	57
Figure 39 Full-text index creation flow for Neo4j database.	58
Figure 40 Common evaluation framework for Neo4j retriever types available.	60
Figure 41 Microsoft SQL server to SQLite database.	61
Figure 42 Overview of process flow of vector index creation in SQLite3 database.	62
Figure 43 SQL agent types and flow overview.	64
Figure 44 Percentage of queries for which LLM either success or fail for proper cypher generation.....	72

List of Tables

Table 1 Common technology stack across experimentation cases.	44
Table 2 Data field and their contextual definition.	44
Table 3 Used LLMs and embedding models with basic info.	45
Table 4 Overview of the feature difference between different retrievers available in GraphRAG python package.	59
Table 5 Overview of main tools for development of agentic systems for SQLite database.	63
Table 6 Local model performance across explicit and implicit query sets (Hybrid retriever).	67
Table 7 Cloud model performance across explicit and implicit query Sets (Hybrid retriever).	68
Table 8 Improvement in success rates (Cloud vs Local).	68
Table 9 Schema-aware pure FAISS vector retrieval performance (cloud models). .	70
Table 10 Improvement in success rates (Pure FAISS vs Hybrid Retriever).	70
Table 11 Retrieval performance on structured RDF documents (Schema-aware prompt).	71
Table 12 Retrieval Performance on Flat Turtle RDF Documents.	71
Table 13 Text2Cypher retriever performance.	73
Table 14 Hybrid vector retriever performance.	73
Table 15 HybridCypherRetriever performance.	74
Table 16 Vector retrieval performance.	74
Table 17 Comparative Performance of GraphRAG Retrievers in Neo4j.	75
Table 18 Simple SQL agent performance.	75
Table 19 Enhanced agent performance.	76
Table 20 Vector enhanced agent performance.	76
Table 21 Overall Comparison of different types of sql agents.	77
Table 22 Key metrics across three case-studies approaches.	79

1 Introduction

This chapter sets the stage for the thesis by outlining the motivation, context, and need of developing RAG systems for manufacturing simulation software. It defines the research problem, objectives, and scope, providing the foundation for the subsequent chapters.

1.1 Background

Manufacturing simulation software, a subset of Computer-Aided Engineering (CAE) tools, enables virtual modelling and optimisation of production systems. Effective use of these tools requires significant user expertise, typically gained through a combination of formal training, documentation, and hands-on practice as depicted in Figure 1. Since the user of the tool is human, the queries that is asked during the use of tool is often in natural language. However, the diversity of information sources and the reliance on human expertise often create bottlenecks in adoption and proficiency.

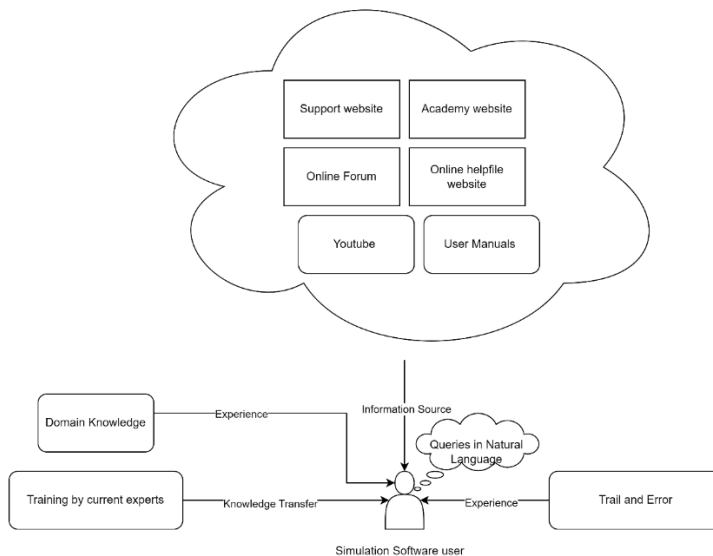


Figure 1 How does a simulation software engineer gain skill in particular software?

To address these challenges, there is growing interest in developing intelligent assistants conceptualized in Figure 2 capable of integrating information from multiple sources and providing context-aware support.

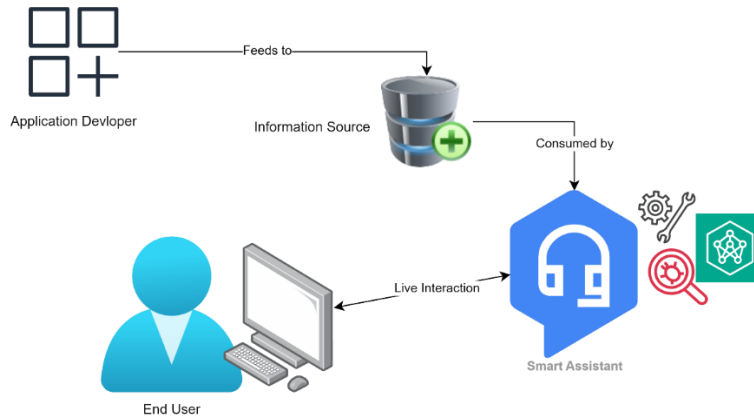


Figure 2 Initial concept of smart assistant.

Recent advances in LLMs have demonstrated the potential for natural language interfaces in technical domains. However, standard LLMs are limited by their lack of access to proprietary or domain-specific data as illustrated in Figure 3.

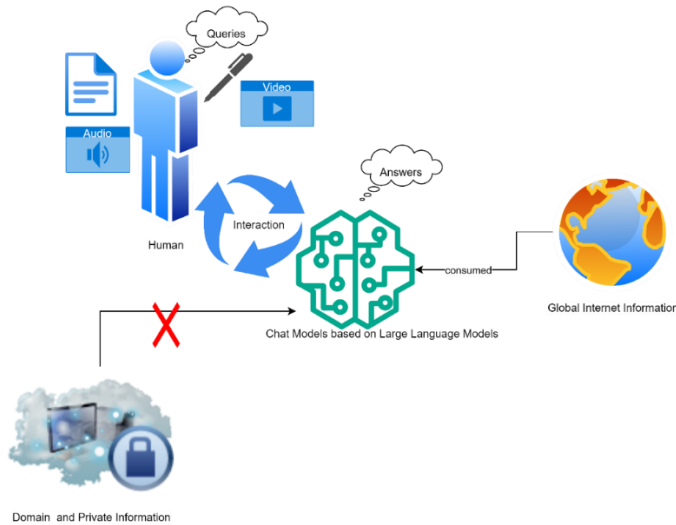


Figure 3 Overview on LLM models.

RAG systems extend LLMs by enabling them to retrieve and incorporate external knowledge, thus improving the accuracy and relevance of their responses to the domain. However, the effectiveness of RAG is fundamentally dependent on the quality of the underlying data sources. This research investigates the impact of data structure and ontologies on the performance of RAG systems in manufacturing simulation, using Visual Components software data as a case study.

Visual Components is a windows-based manufacturing simulation software widely used in industry for its extensive library of digital components

and intuitive “drag-and-drop” interface. It supports key simulation areas such as 3D layout planning, material flow, logistics, robotics, virtual commissioning, and offline robot programming. For advanced users, the software also provides APIs in python and .NET, enabling customisation and integration with other engineering tools. The core structured data source within Visual Components is its eCatalog a relational database of ready to use components. In contrast, the remaining information required for users to master the software is found in unstructured formats, such as user manuals, help files, tutorials, and similar resources.

1.2 Research problem

While RAG systems have demonstrated promise in enhancing LLM capabilities, their effectiveness in manufacturing simulation environments is fundamentally influenced by the structure and representation of the underlying data. This thesis addresses the following research questions:

- 1) How do different formats of data representation (e.g., JSON, RDF, knowledge graphs, relational table) , affect the efficiency and accuracy of information retrieval in RAG systems within the context of manufacturing simulation?
- 2) What is the role of ontologies in improving information retrieval, and what best practices can be established for integrating ontologies with RAG architectures?
- 3) What are the practical benefits and limitations of deploying RAG systems in data from manufacturing simulation environments, particularly regarding retrieval accuracy, system reliability, and engineering complexity?

By systematically investigating these questions, the thesis aims to clarify the interplay between data modelling, retrieval strategies, and LLMs capabilities in real-world engineering applications.

1.3 Objectives

The objectives of this thesis are to:

- 1) Empirically evaluate how different approaches to structured data organisation and association impact the performance of RAG systems in manufacturing simulation.
- 2) To identify and recommend best practices for data modelling, retrieval architecture selection, and prompt engineering in RAG systems for manufacturing simulation software, with a particular focus on comparing the effectiveness of vector store, knowledge graph, and relational table-based retrieval approaches.

1.4 Scope

This thesis focuses on the application of pre-trained LLMs in RAG systems, without fine-tuning the base models. The research centres on proprietary eCatalog metadata from Visual Components, represented in structured formats such as JSON, RDF, knowledge graphs and relational tables. The study is experimental and comparative, employing both local and cloud-based LLM and embedding models, and systematically evaluating their performance across three distinct data storage paradigms: FAISS vector stores, Neo4j knowledge graphs, and SQLite3 relational databases. The evaluation framework emphasises retrieval accuracy, context-awareness, and engineering trade-offs, with a particular focus on explicit versus implicit query handling.

1.5 Thesis outline

The remainder of this thesis is organised as follows:

Chapter 2: Literature review

This chapter reviews the relevant literature on manufacturing simulation software, data and knowledge structures in engineering, information retrieval models, semantic technologies, and the evolution of artificial intelligence. It focuses particularly on LLMs and RAG systems.

Chapter 3: Research material and methods

This chapter describes the research design, data sources, data preparation pipeline, and the experimental frameworks used to evaluate the impact of ontologies and data structures on RAG performance in manufacturing simulation.

Chapter 4: Results

This chapter presents the results of the three case studies, comparing the effectiveness of different data structures and retrieval architectures, and analysing the performance of various RAG applications.

Chapter 5: Discussion and conclusions

This chapter interprets the results, discusses practical implications for manufacturing simulation, answers the research questions, highlights academic and practical contributions, and outlines the limitations of the study and directions for future research.

2 Literature review

This chapter reviews key concepts and prior research relevant to this study. It covers manufacturing simulation software, the transformation from data to knowledge in engineering, classical and modern information retrieval models, semantic technologies, and the evolution of artificial intelligence, with a focus on LLMs and RAG systems.

2.1 Manufacturing simulation software

Manufacturing simulation refers to the use of computational models to represent and analyse manufacturing systems, processes, and workflows. As defined by Nassehi and Urgo(2019), simulation is “the dynamic observation of an abstract model of a system through time with particular attention to the system’s key attributes.” In the manufacturing domain, simulation enables engineers to visualise entire factories, individual equipment, and supply chains, supporting both discrete event simulations (DES) and continuous time simulation systems.

Visual Components, a commercially available software, emphasises visual simulations and follows the principles of DES (Visual Components, 2025). In DES, a system’s operation is represented as a sequence of discrete events, each marking a change of state at a specific moment in time.

By leveraging manufacturing simulation tools, engineers can transform raw data into actionable information and knowledge. These tools facilitate the conceptualisation, commissioning, performance analysis, and optimisation of manufacturing assets, forming the foundation for data-driven decision-making in modern production environments.

2.2 Data, information and knowledge in engineering

A clear distinction between data, information, and knowledge is fundamental in engineering, especially within manufacturing simulation and the deployment of advanced AI tools. The Data-Information-Knowledge-Wisdom (DIKW) hierarchy (Rowley, 2007) describes how raw facts are transformed into actionable insights and decisions.

- **Data** are raw, unprocessed facts such as sensor readings or lists of machine manufacturers, lacking context or meaning on their own (Ackoff, 1989).
- **Information** arises when data are organised or structured to provide context, answering questions like who, what, where, and when (Dav-enport Thomas and Prusak Laurence, 1998). For example, grouping machine manufacturers by product type enables meaningful comparisons.
- **Knowledge** is created when information is contextualised and applied through experience and reasoning, enabling decision making

and problem solving. This can be explicit: easily articulated and shared or tacit: personal and harder to formalise (Nonaka Ikujiro and Hirotaka Takeuchi, 1995) .

This progression is illustrated in Figure 4.

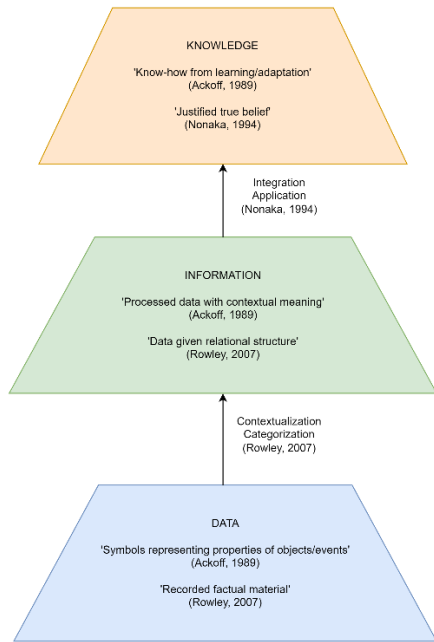


Figure 4 How data transforms into knowledge?

In practice, simulation software accelerates the transformation from raw data to actionable knowledge. For instance, raw sensor data (data) are processed by simulation models to reveal system states (information), which engineers then use to optimise processes (knowledge). The goal is to support informed decision making and operational excellence in manufacturing environments.

Figure 5 illustrates how raw data is systematically transformed into actionable wisdom in 3D manufacturing simulation. The process begins with unprocessed data, such as CAD models, process parameters, sensor streams, and error logs. This data is organised into information: ready-made simulation components, layout configurations, and process flow diagrams which provides meaningful insights into system operation. Knowledge is then generated through optimisation models, robot path planning, and bottleneck prediction tools, supporting informed decision-making. At the wisdom level, advanced applications like digital twin validation, OEE optimisation, and CAPEX scenario modelling guide strategic planning and operational excellence. Key simulation metrics, such as OEE, energy consumption, and throughput rate, reflect the culmination of this hierarchy and highlight the importance of structured data processing for effective decision making in industrial environments.

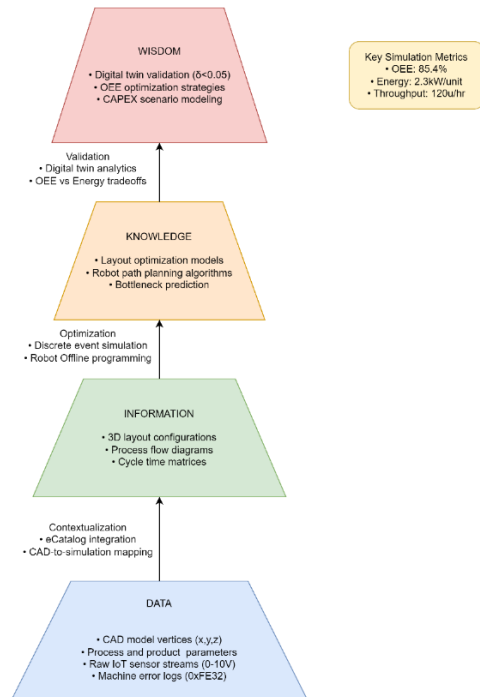


Figure 5 Application of DIKW(Rowley, 2007) hierarchy in 3D manufacturing simulation.

The transformation from data to information involves contextualising data, organising it into categories, performing calculations, correcting errors, and condensing it for clarity. Moving from information to knowledge is more complex, requiring integration with existing understanding, interpretation through conceptual frameworks, practical application, and continuous learning from outcomes.

Understanding how these layers interact within simulation software is crucial for leveraging machine learning and artificial intelligence. The aim is to reduce the time and effort required to move from raw data to actionable wisdom, as depicted in Figure 6.

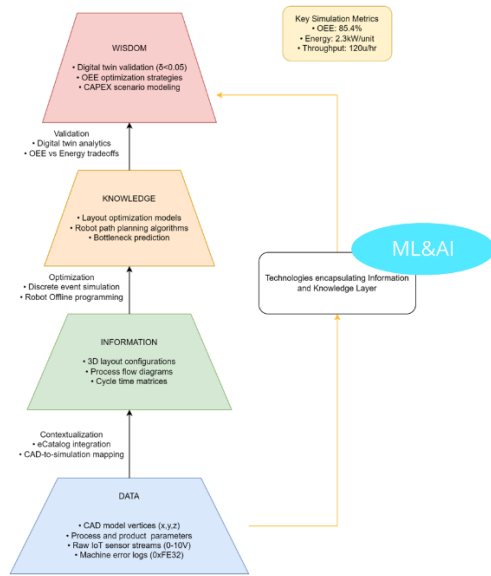


Figure 6 How ML/AI can abstract information extraction and knowledge creation?

Knowledge in engineering exists in multiple forms, and it is important to appreciate both the types of knowledge and how they relate to users of manufacturing simulation software.

2.2.1 Knowledge types in engineering simulation and their user correlation

A comprehensive understanding of knowledge types is essential for developing and using simulation software in engineering, as in this domain, knowledge is not monolithic; rather, it encompasses several distinct yet interrelated categories, each contributing uniquely to user proficiency and system advancement as illustrated in Figure 7.

Explicit knowledge includes documented facts and instructions, supporting onboarding and consistent tool use. Implicit knowledge arises from practical application, enabling users to interpret nuanced scenarios, while tacit knowledge is gained through personal experience and intuition. Procedural knowledge covers step-by-step processes and workflows, and declarative knowledge involves understanding theoretical principles. A posteriori knowledge is derived from empirical validation, ensuring models reflect real-world behaviour, whereas a priori knowledge is based on logical reasoning and theory, guiding initial model setup.

These types interact dynamically within simulation platforms and are embodied by users at different expertise levels: novices rely on formal and foundational knowledge, intermediates develop experiential knowledge, and experts leverage tacit and empirical insights for advanced decision-making.

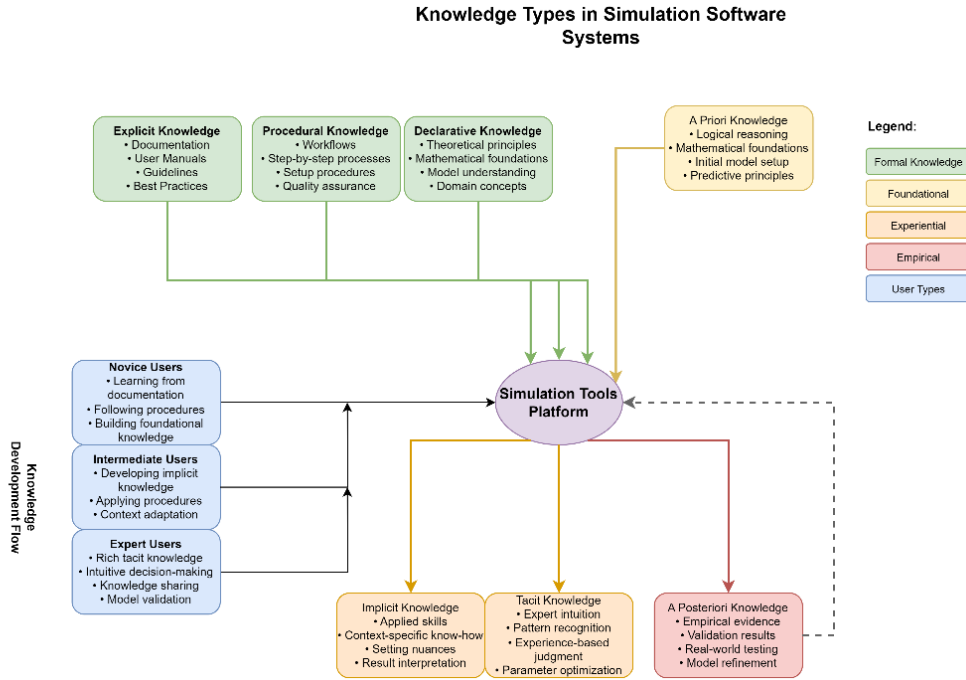


Figure 7 Knowledge types and user progression in simulation software systems.

Recognising these categories is crucial for designing AI assistants that support learning, knowledge transfer, and problem-solving in simulation environments, forming a foundation for research into knowledge representation and retrieval.

2.3 Information retrieval and semantic technologies

Information retrieval (IR) underpins the extraction of relevant data from large and complex datasets, a challenge central to modern engineering and simulation environments. This section reviews classical and contemporary IR models, as well as semantic technologies, to contextualise the technical landscape relevant to this research.

2.3.1 Classical IR models

The Boolean model, one of the earliest IR approaches, represents documents and queries as sets of terms and applies Boolean logic (AND, OR, NOT) to determine relevance. While simple, it does not support ranking by degree of relevance (Robertson and Jones, 1976).

The vector space model (Salton et al., 1975) encodes documents and queries as vectors in a multi-dimensional space, enabling partial matching and ranking based on similarity measures such as cosine similarity.

Probabilistic models estimate the likelihood that a document is relevant to a query, incorporating statistical measures such as term frequency (Robertson and Jones, 1976).

Modern IR approaches, including PageRank (Brin and Page, 1998) and learning-to-rank algorithms (Joachims, 2002), leverage link analysis and user feedback to improve retrieval relevance. Recent advances in machine learning and deep learning have further enhanced IR performance by enabling models to learn complex semantic patterns from large datasets (Bhaskar and Craswell, 2017).

2.3.2 Semantic technologies in information retrieval

Semantic technologies, such as ontologies, the Resource Description Framework (RDF), and knowledge graphs, enable IR systems to capture and utilise the meaning and relationships within data. Ontologies formally define domain concepts and relationships, improving query understanding and retrieval precision (Guarino et al., 2009). RDF represents data as subject-predicate-object triples, facilitating interoperability and integration across sources (Hitzler et al., 2010). Knowledge graphs provide a structured, interconnected representation of entities and their attributes, supporting advanced reasoning and context-aware retrieval (Hogan et al., 2022).

The development of the web and information retrieval and organization techniques created abundance of data in the world. This helped acceleration of studying how machines can learn from the data and shorten the cycle of data to knowledge which bring us now in the age of artificial intelligence.

2.4 Evolution of artificial intelligence

The evolution of artificial intelligence is characterized by distinct developmental phases. There are three primary eras of AI development as illustrated in the provided timeline in Figure 8: the Machine Learning Era (Pre-2000s), the Deep Learning Era (2000s-2010s), and the Foundation Models Era (2020s-Present) (Bommasani et al., 2022).

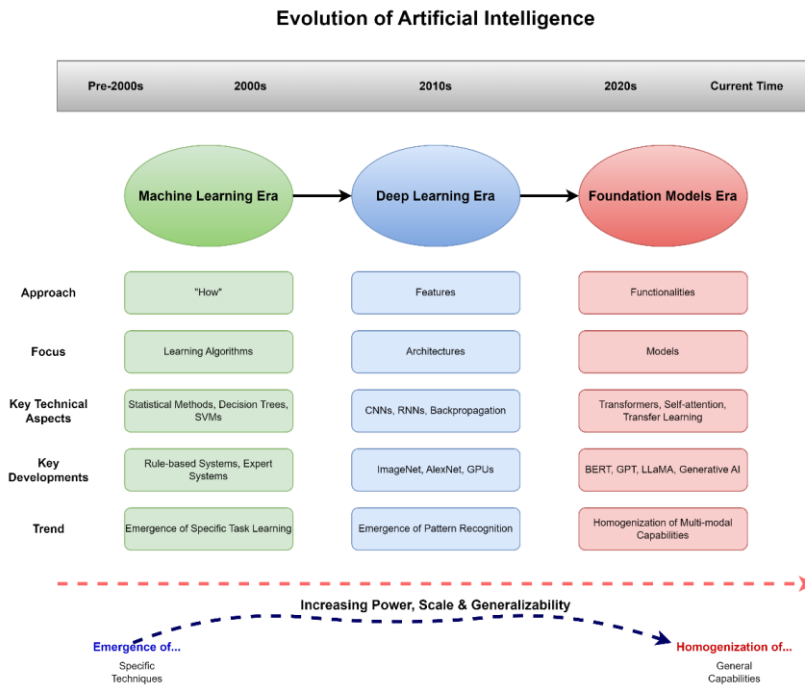


Figure 8 Evolution of Artificial Intelligence (Bommasani et al., 2022).

Machine learning era (Pre-2000s):

The early phase of AI was characterised by the development of rule-based systems and statistical learning algorithms. Researchers focused on methods such as decision trees and support vector machines (SVMs), which enabled computers to perform specific tasks by following explicit rules or learning from labelled data. Rule-based systems, including expert systems, replicated human decision-making through “IF-THEN” logic, but were limited to narrow, domain-specific applications.

Deep learning era (2000s-2010s):

The advent of deep learning marked a paradigm shift from algorithmic rule-following to data-driven architectural innovation. Neural network structures, particularly convolutional neural networks (CNNs) and recurrent neural networks (RNNs), enabled significant breakthroughs in pattern recognition and computer vision. The introduction of AlexNet (Krizhevsky et al., 2017) in the ImageNet Large Scale Visual Recognition Challenge demonstrated the power of deep learning, achieving unprecedented accuracy in image classification. This era also saw the widespread adoption of graphics processing units (GPUs), which provided the computational resources necessary for training large-scale neural networks.

Foundation models era (2020s-Present):

The most recent phase is defined by the emergence of foundation models: large-scale, pre-trained neural networks capable of generalising across

diverse tasks. These models, such as BERT (Devlin et al., 2019), GPT (Brown et al., 2020), and LLaMA (Touvron et al., 2023), are built on the transformer architecture (Vaswani et al., 2017), which employs self-attention mechanisms for efficient context modelling. Foundation models are trained on vast datasets and contain billions or even trillions of parameters, enabling zero-shot and few-shot learning. The current era is also characterised by the rise of multimodal AI systems, which integrate text, images, audio, and video, and by the “homogenisation” of capabilities across different modalities. (Bommasani et al., 2022)

This progression from rule-based systems to neural networks and foundation models reflects broader trends in computational power, data availability, and algorithmic sophistication. The transition has enabled AI systems to evolve from narrow, task-specific tools to general-purpose platforms capable of addressing complex, real-world problems.

2.4.1 Technology stack for generative artificial intelligence

The generative AI technology stack comprises several interdependent layers, each contributing to the development, deployment, and maintenance of modern AI systems. Figure 9 provides an overview of these layers and their relationships. The generative AI technology stack is built on a hardware infrastructure layer comprising specialised processors (GPUs, TPUs, IPUs, NPUs), scalable cloud platforms (AWS, Azure, Google Cloud), and distributed computing architectures with high-bandwidth memory and advanced storage solutions to support large-scale model training and inference (Zhang et al., 2025). The model training and optimisation layer leverages distributed training (data, model, and pipeline parallelism), frameworks like DeepSpeed and Megatron-LM, and techniques such as mixed-precision training, knowledge distillation, and quantisation to efficiently develop and refine models (Smith et al., 2022). At its core, the foundation models layer includes large transformer-based models (e.g., gpt-4, Claude-3, LLaMA, Mistral) trained on vast datasets, supporting both unimodal and multimodal capabilities, with emergent properties arising from scale (Naveed et al., 2024). The adaptation and orchestration layer enables customisation through fine-tuning (RLHF, DPO, PEFT/LoRA), workflow automation, and inference optimisation (quantisation, caching, batching), ensuring models are efficiently deployed and tailored to specific tasks. The application and interface layer provides frameworks and tools for building user facing AI applications such as copilots, chatbots, and domain-specific assistants via APIs, plugins, and agentic workflows that support complex reasoning and integration. Over-arching all layers, the safety, ethics, and governance layer implements risk mitigation, bias detection, transparency, and regulatory compliance (e.g., EU AI Act, GDPR), while cross-cutting concerns like interpretability, security, and fairness permeate the entire stack, making robust architecture and

responsible deployment essential for effective generative AI systems (Bommasani et al., 2022; Naveed et al., 2024).

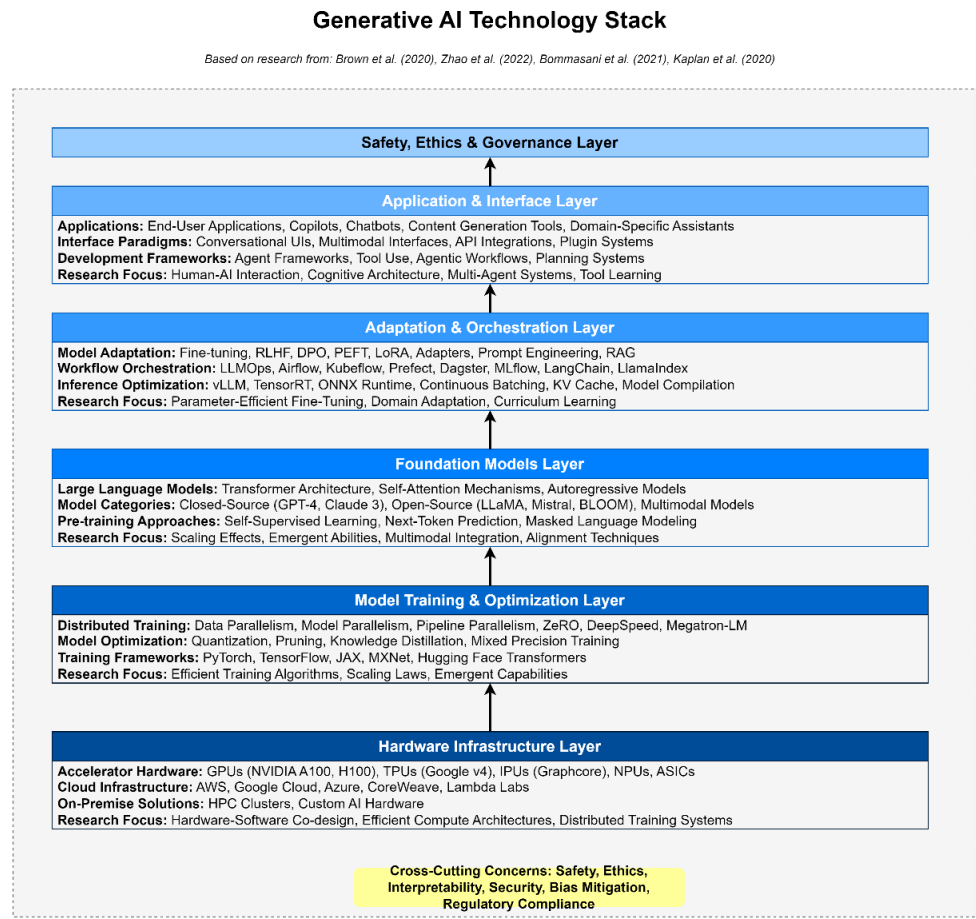


Figure 9 Generative AI technology stack.

In this thesis, we are particularly interested in one type of foundation models called LLMs.

2.4.2 LLMs

LLMs are foundational models developed using deep learning techniques on massive, diverse text corpora. Their key innovation over previous natural language processing approaches is the ability to understand and generate human-like text, enabling a wide range of applications in language understanding, generation, and reasoning (Bommasani et al., 2022)

The public release of ChatGPT by OpenAI marked a pivotal moment, rapidly reaching 100 million users and catalysing substantial investment and research activity. In 2024 alone, over 49,000 research papers were published on LLMs, reflecting the rapid growth and interest in this domain(Naveed et al., 2024). This thesis focuses on the use of pretrained LLMs, without

retraining or fine-tuning, and considers both open-source and closed-source models.

Closed source LLMs, or proprietary models, are developed by major technology companies and are characterised by restricted access to their internal architecture, training methods, and implementation details. These models are typically accessed via controlled APIs or web-based platforms, with prominent examples including OpenAI’s GPT series: gpt-3.5, gpt-4, gpt-4o (OpenAI, 2025), Anthropic’s Claude models: Claude 3, Claude 3.5 Sonnet (Anthropic, 2025), and Google’s Gemini family: Gemini 1.0, Gemini 1.5 Pro (Google DeepMind, 2025). While closed source models benefit from significant computational resources, large-scale training datasets, and rigorous safety alignment, their proprietary nature raises concerns about transparency, reproducibility, and potential data contamination issues that are particularly relevant in academic research (Bender et al., 2021).

In contrast, open source LLMs provide full access to model weights, training code, and architectural specifications. Leading examples include Meta’s LLaMA series: LLaMA 2, LLaMA 3, LLaMA 3.1 (Touvron et al., 2023), Mistral AI’s models, Google’s T5 and Gemma (Google DeepMind, 2025b), and community-driven projects such as LLM360 (Liu et al., 2023). Open-source models enable comprehensive architectural analysis, support privacy-preserving applications, and allow for fine-tuning and adaptation to specialised domains (Touvron et al., 2023). Their transparency fosters reproducibility and encourages broader research participation (Bender et al., 2021).

When a foundational model like LLMs are trained there are pre-training and post-training processes which are described in section 2.4.3 and section 2.5 respectively.

2.4.3 Pre-training process of LLMs

The pre-training of LLMs is a multi-stage pipeline as illustrated in Figure 10 that transforms raw textual data into powerful language understanding systems. The process begins with large-scale data collection from diverse sources, followed by rigorous preprocessing: filtering, deduplication, cleaning, normalisation, and privacy reduction to ensure high-quality training material (Ostendorff et al., 2024; Sachdeva et al., 2024). Tokenisation, often using byte-pair encoding (BPE) (Gage, 1994), converts text into numerical representations suitable for neural network processing.

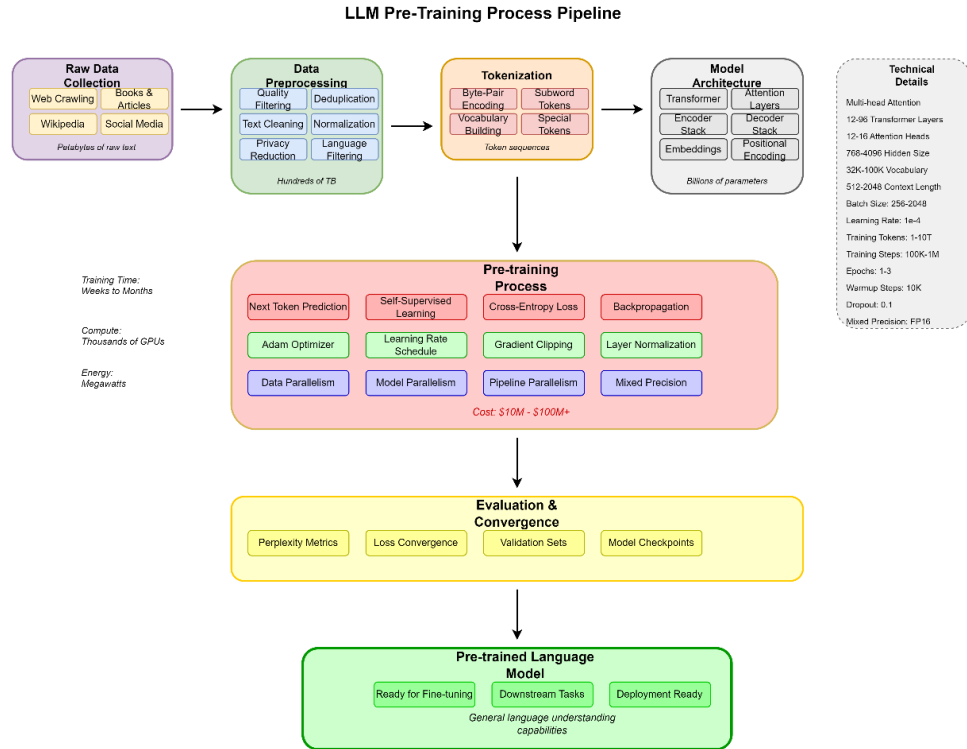


Figure 10 LLM pre-training process pipeline.

LLMs are built on the transformer architecture (Vaswani et al., 2017), which uses self-attention and multi-head attention mechanisms to model complex relationships in sequential data. Variants include encoder-only example BERT (Devlin et al., 2019), decoder-only example GPT (Brown et al., 2020), and encoder-decoder example T5 (Raffel et al., 2023) models, each optimised for different tasks. Encoder-only models excel at understanding and classification tasks, decoder-only models are effective for generative tasks, and encoder-decoder models offer balanced capabilities for both understanding and generation.

Pre-training is computationally intensive, involving next-token prediction via self-supervised learning, cross-entropy loss, and backpropagation. Optimisation strategies include Adam optimisers (Kingma and Ba, 2017), learning rate scheduling, gradient clipping, and parallelisation (data, model, and pipeline) to manage the demands of large-scale training. Evaluation uses metrics such as perplexity and loss convergence, with regular checkpointing to ensure stability and prevent overfitting. The resulting pre-trained model provides general language understanding, ready for fine-tuning on specific tasks (Sachdeva et al., 2024; Vaswani et al., 2017).

Resource requirements for LLM pre-training are substantial, often involving weeks to months of training, thousands of GPUs, and significant power consumption. For example, training Google’s PaLM model (Chowdhery et al., 2022) required over \$10 million and two months of supercomputing.

Technical specifications typically include dozens of transformer layers, thousands of attention heads, and context lengths up to several thousand tokens.

Advantages of LLMs:

LLMs have achieved human-level performance in a range of natural language understanding (NLU) and generation (NLG) tasks, surpassing previous benchmarks. Their general-purpose nature enables flexible application across domains, supporting context-aware and user-friendly solutions. For example, tools like GitHub Copilot facilitate code generation and assist both novice and expert users by automating routine tasks.

Limiting factors in LLMs:

Computational challenges:

LLMs incur significant costs, often amounting to millions of dollars. For instance, retraining U-PaLM from PALM for 120 hours costs approximately \$250,000 (Naveed et al., 2024). Inference is also resource-intensive; deploying gpt-3 with 175 billion parameters requires at least five 80GB A100 GPUs and 350GB of memory for FP16 format (Xiao et al., 2023). Most standard devices are not equipped for such workloads, necessitating specialised hardware with high end NPUs, GPUs or reliance on cloud computing infrastructure. This creates additional challenges for desktop-based applications, which require robust connectivity for cloud resources or advanced local architectures to exploit available CPU, GPU, or NPU capabilities.

Limited Knowledge:

LLMs are trained on large public datasets, but the specific training data is rarely disclosed. As a result, these models lack access to enterprise or proprietary information, and their knowledge does not update dynamically with new developments. To incorporate new information, retraining is required; for example, a model released in 2022 cannot answer questions about events or discoveries from 2025. The effectiveness of an LLM is fundamentally limited by the quality and scope of its training data.

Reasoning and planning:

LLMs are primarily designed for NLG and NLU, not for robust reasoning or planning. While some proprietary models like OpenAI's o1 claim enhanced reasoning capabilities, these claims cannot be verified due to undisclosed architectures. Although improvements have been observed in specific planning benchmarks like PLANBENCH, LLMs generally lack consistent and reliable reasoning and planning abilities (Valmeekam et al., 2024).

Hallucinations, Safety and Controllability:

LLMs can generate convincing but inaccurate or irrelevant outputs, a phenomenon known as hallucination (Bang et al., 2023; Huang et al., 2023). Additionally, these models may produce misleading or harmful content (Shaikh et al., 2023), raising concerns about safety and controllability in enterprise applications. Careful monitoring and robust control mechanisms are

essential to ensure the integrity and reliability of LLM generated responses, especially in contexts involving critical systems or external components.

Applications built on foundational LLMs can be broadly categorised as either reactive or proactive. Reactive applications, such as chatbots and virtual assistants, operate in a stateless manner, responding to user prompts without maintaining long-term context or engaging in autonomous planning (Bommasani et al., 2022). In contrast, proactive applications often referred to as LLM-based agents are capable of goal-directed behaviour, planning, reasoning, and interacting with external tools or environments to achieve complex objectives with minimal user intervention (X. Liu et al., 2025; Yao et al., 2023).

This thesis limit to working with a pretrained model and stored information that needs retrieval, so next chapter describes the foundations of RAG.

2.5 RAG foundations

RAG is a class of LLM applications that enhance response accuracy and contextual relevance by dynamically retrieving information from external sources, such as enterprise databases or domain-specific repositories, during inference. This approach is particularly valuable in specialised domains like manufacturing simulation, where up-to-date and context-specific data are essential. Formally, a RAG application can be expressed as a mapping function:

$$f : Q \xrightarrow{D} A$$

Where:

- Q denotes the user query,
- D represents the external data source,
- A is the generated answer,
- f is the LLM application that maps Q to A using D as context

This mathematical representation captures the essence of RAG systems, where the function “f” leverages external data D to transform user queries Q into accurate answers A (S. Zhao et al., 2024). By incorporating external knowledge at inference time, RAG systems overcome the inherent limitations of LLMs that rely solely on static, pre-trained data, ensuring responses remain current and relevant.

2.5.1 RAG system architecture and core components

The fundamental architecture of a RAG system integrates two primary components: the retriever and the generator. Figure 11 illustrates the generic architecture of a RAG system, where user queries serve as input to both components, creating a synergistic information flow.

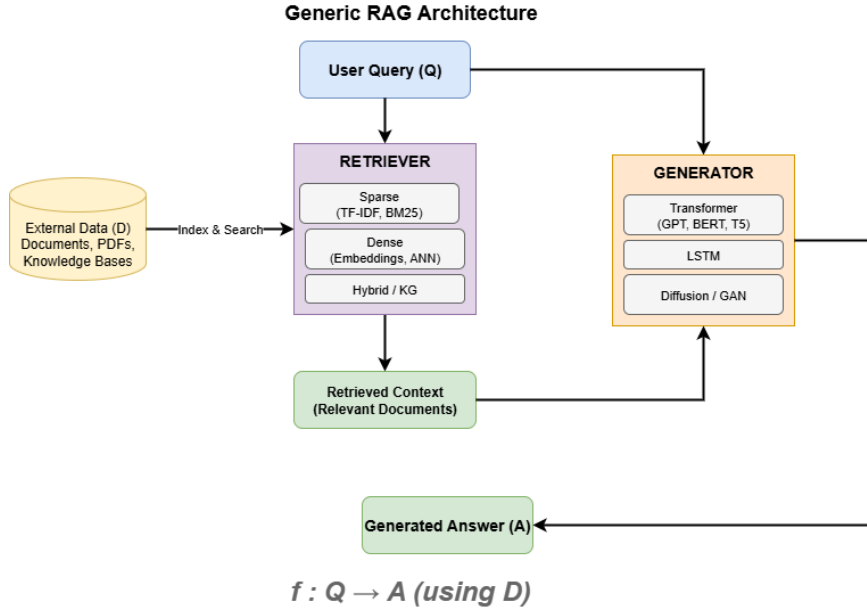


Figure 11 Generic RAG architecture (S. Zhao et al., 2024).

The retriever is responsible for identifying and extracting relevant information from external sources, which may include structured databases, document repositories, or knowledge graphs. This process ensures that the system can access the most pertinent and current data in response to a user’s query. The retrieved information is then passed to the generator, which synthesises a coherent and contextually appropriate response by integrating the user’s query with the retrieved content. This separation of retrieval and generation not only grounds the output in factual content reducing the risk of hallucinations but also allows the system to be updated with new information without the need for retraining the underlying language model, supporting both scalability and maintainability (S. Zhao et al., 2024).

The retriever component can be implemented using several approaches. Sparse retrievers, such as those based on TF-IDF (Salton et al., 1975) or BM25 (Robertson and Jones, 1976), rely on lexical matching and are computationally efficient, excelling at exact keyword searches but limited in handling semantic similarity. Dense retrievers, by contrast, use neural networks to encode queries and documents into dense vector embeddings, enabling semantic search and effective handling of paraphrased or conceptually similar queries. Techniques such as Dense Passage Retrieval (DPR) (Karpukhin et al., 2020) and Approximate Nearest Neighbour (ANN) based search such as HNSW: Hierarchical Navigable Small World (Malkov & Yashunin, 2020), FAISS: Facebook AI Similarity Search (Johnson et al., 2021), LSH: Locality-Sensitive Hashing (Indyk & Motwani, 1998) are commonly employed in this context. Hybrid retrieval strategies, which combine sparse and dense

methods, are increasingly adopted to leverage the strengths of both approaches (S. Zhao et al., 2024). Additionally, knowledge graph-based retrieval (Hogan et al., 2022) and named entity recognition (NER) (S. Zhao et al., 2024) can be integrated to support complex, multi-hop, and entity-centric queries.

The generator component typically employs advanced neural architectures. Transformer-based LLMs models like GPT, BERT, and their variants represent the most common choice for RAG generators (Brown et al., 2020; Devlin et al., 2019; Vaswani et al., 2017). Other architectures, including Long Short-Term Memory (LSTM) networks (Hochreiter and Schmidhuber, 1997), diffusion models, and Generative Adversarial Networks (GAN), may be utilised depending on the specific requirements of the application, such as the need for sequential memory or multi-modal outputs (P. Zhao et al., 2024; S. Zhao et al., 2024). The choice of generator is determined by factors such as the desired output modality, computational resources, and quality expectations.

2.5.2 Data types and management in RAG

An effective RAG system must handle a variety of data types, each requiring specific processing strategies to optimise retrieval accuracy and system performance. Data are mainly of two types structured and unstructured.

Structured data follows standardised formats with well-defined schemas, such as JSON, XML, CSV, or SQL databases. These formats facilitate programmatic processing and querying, allowing direct transformation into natural language descriptions suitable for embedding and retrieval.

Unstructured data, by contrast, is found in sources that lack a standard format, requiring specialised parsing and extraction techniques. Examples include PDF documents (which may require OCR or text extraction), plain text files, HTML web pages, DOCX files, and multimedia content such as images and videos, all of which require multi-modal processing approaches. Processing pipelines for unstructured data typically involve data gathering, extraction, cleaning, chunking, embedding generation, and storage in vector databases (S. Zhao et al., 2024).

Chunking the process of dividing documents into smaller, retrievable segments is critical for RAG performance (Tanyildiz et al., 2024). The choice of chunking strategy significantly impacts retrieval quality and downstream generation:

Fixed-Size Chunking: Divides text into segments of predetermined size (e.g., 512 tokens). This approach is simple to implement but may break semantic units; typically, some overlap is included between chunks to preserve context (Tanyildiz et al., 2024).

Semantic Chunking: Respects document structure (e.g., paragraphs, sections), preserving semantic coherence within chunks. This method is more complex but generally yields better retrieval quality (Tanyildiz et al., 2024).

Recursive Chunking: Attempts to split on natural boundaries (e.g., paragraphs, then sentences), falling back to character-level splitting if necessary. This balances semantic coherence with size constraints (Tanyildiz et al., 2024).

The chunk size should align with the embedding model's context window and overlap between chunks helps preserve boundary context. Metadata such as section headers and document titles should also be preserved for optimal retrieval (Tanyildiz et al., 2024).

Embeddings are dense vector representations that capture semantic meaning. Modern embedding models (e.g., OpenAI's text-embedding-3, Cohere's embed-v3, Sentence-BERT) convert text into high-dimensional vectors (typically 384-1536 dimensions), ensuring that semantic similarity is reflected in vector proximity and that dimensionality is consistent and task-optimised (Z. Liu et al., 2020). Figure 12 shows an example of embedding representation of text.

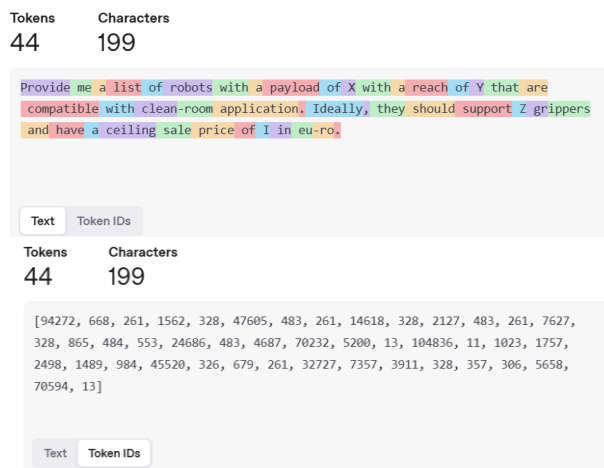


Figure 12 Example of embedding representation of text.

Vector databases provide storage and retrieval of embeddings through optimised indexing structures. Popular options include Pinecone, Chroma, FAISS, Weaviate, and Qdrant (Huerga-Pérez et al., 2025; Johnson et al., 2021). Selection criteria for vector databases include scale requirements (millions vs. billions of vectors), deployment preferences (cloud-managed vs. self-hosted), performance needs (latency and throughput), budget constraints, and integration ecosystem compatibility (Huerga-Pérez et al., 2025).

2.5.3 Query taxonomy in RAG systems

Queries in RAG systems vary in complexity and the depth of reasoning required. Understanding these query types is essential for designing effective retrieval strategies. Zhao et al., (2024) have categorized queries into four levels as follows:

1) Explicit fact Queries

These seek information directly present in the dataset, requiring no additional reasoning.

Example: "Where is the Visual Components headquarter located?"

The answer is retrieved verbatim from the relevant document chunk.

2) Implicit fact Queries

These require basic reasoning or combining multiple facts from the dataset.

Example: "What is the yearly GDP of the country where Visual Components Headquarter is located?"

The system must first identify the headquarters location (Finland), then retrieve Finland's GDP data.

3) Interpretable Rationale Queries

These demand domain-specific reasoning, considering implicit knowledge or safety constraints.

Example: "How can I control a real robot system using simulation software?"

The system must understand simulation domain risks, such as unmodelled physical obstacles, and provide appropriate safety guidance.

4) Hidden Rationale Queries

These require pattern recognition and contextual understanding beyond explicit documentation.

Example: In customer support, identifying patterns in responses based on region or customer type.

The system must infer answers from trends and contextual data.

Understanding these query hierarchies enables RAG system designers to implement appropriate retrieval strategies as described in next section.

2.5.4 Query complexity framework and RAG techniques

The relationship between query complexity and suitable RAG techniques can be systematically understood through a framework that maps query types to optimal retrieval and generation strategies. As illustrated in Figure 13, this framework aligns five levels of RAG approaches with both query complexity and Bloom's Cognitive Taxonomy (S. Zhao et al., 2024).

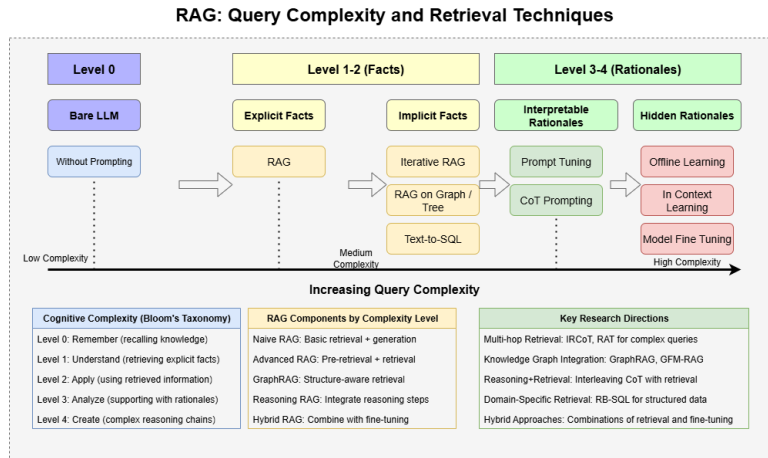


Figure 13 RAG query complexity and retrieval techniques framework (based on S. Zhao et al., 2024).

At the foundational level (Level 0), bare LLMs address simple queries without retrieval augmentation, corresponding to the "Remember" level of Bloom's Taxonomy. Intermediate levels (1-2) encompass explicit and implicit fact queries, addressed through naive and advanced RAG techniques. More complex queries (Levels 3-4) require reasoning and rationale, addressed through prompt tuning, chain-of-thought prompting, and model fine-tuning.

For explicit fact queries, Naive RAG implements basic retrieval and generation, while implicit fact queries benefit from iterative retrieval, graph/tree-based approaches, and text to SQL conversion. Interpretable rationale queries are addressed through prompt tuning and chain-of-thought prompting, whereas hidden rationale queries require offline learning, in-context learning, and model fine-tuning.

Key research directions include multi-hop retrieval, knowledge graph integration, and hybrid approaches, which enable systems to handle increasingly complex queries (Bommasani et al., 2022; Sun et al., 2024; Tanyildiz et al., 2024; S. Zhao et al., 2024).

This framework provides practitioners with a practical roadmap for selecting appropriate RAG architectures based on anticipated query complexity, supporting effective system design and resource allocation.

2.5.5 Static versus dynamic RAG applications

RAG systems can be classified based on how they manage updates to their underlying knowledge sources. Static RAG applications maintain a fixed snapshot of the knowledge base, requiring manual intervention to refresh content when source data changes. This approach is architecturally simple and suited to domains with stable information, such as archival search or reference systems. In contrast, dynamic RAG applications automatically detect and incorporate changes in source data, enabling real time or near real time updates. These systems provide more current and contextually accurate

responses but require greater engineering complexity and operational overhead (Sun et al., 2024; S. Zhao et al., 2024).

2.5.6 Orchestration frameworks for LLM service implementation

Developing production grade RAG applications requires orchestration frameworks that simplify the integration of LLMs with diverse data sources and retrieval mechanisms. Two leading frameworks in this domain are LangChain and Semantic Kernel (LangChain, 2025b; Microsoft, 2025d).

LangChain is an open-source framework designed for building modular LLM-powered applications. It offers components such as chains (sequential workflows), agents (autonomous decision modules), document loaders, text splitters, and vector store abstractions. LangChain's extensibility and integration capabilities make it well-suited for rapid prototyping and experimentation in both academic and industrial contexts (LangChain, 2025b).

Semantic Kernel, developed by Microsoft, focuses on enterprise integration and task automation. It features skills-based modular design, persistent memory for conversational context, advanced planning, and enterprise-grade security. Semantic Kernel is natively integrated with Azure services, supporting seamless deployment within the Microsoft ecosystem (Microsoft, 2025d).

While both frameworks enable sophisticated RAG applications, LangChain is often preferred for research projects due to its flexibility and used in this thesis.

2.5.7 Prompt engineering for RAG systems

Prompt engineering is essential for maximising the effectiveness of LLM-based RAG systems, as prompt design directly impacts the quality and factual accuracy of generated responses. (Brown et al., 2020; Wei et al., 2023). Several key prompting strategies are commonly used:

- 1) **Zero-Shot Prompting**: The model is instructed to perform a task without explicit examples, relying solely on its pre-trained knowledge. This is effective for generic, well-understood tasks. (Brown et al., 2020).

Example:

Given the following context, answer the user's question. Context: {retrieved_context} Question: {user_query} Answer:

- 2) **Few-Shot Prompting**: The model is provided with a small number of input-output examples to guide its response format and style, improving consistency and accuracy, especially in domain-specific contexts. (Brown et al., 2020).

Example:

Context: The company was founded in 1998 in Helsinki, Finland.
Question: When was the company founded?
Answer: The company was founded in 1998.
Context: {retrieved_context}
Question: {user_query}
Answer:

- 3) **Cue-Based Prompting:** Cue-based prompting uses explicit prefixes or cues to direct the model's output, reducing irrelevant or verbose responses. For instance, starting a prompt with "Based on the context provided, the answer is:" encourages concise, direct answers.
- 4) **Chain-of-Thought(CoT) Prompting:** CoT prompting encourages the model to articulate intermediate reasoning steps. This strategy has empirically shown to improve model performance on tasks requiring multi-step reasoning and interpretable rationale (Wei et al., 2023).

Example:

Let's solve this step by step:
1. Identify the key information from the context.
2. Determine the reasoning required.
3. Provide the answer based on this reasoning.

2.6 Applied research on LLMs, RAG, and ontologies in manufacturing simulation

Recent research demonstrates that integrating LLMs, RAG, and ontologies can streamline and enhance manufacturing simulation processes. For example, Makatura et al.,(2024) showed that LLMs can automate the translation of design requirements into engineering tasks, supporting CAD engineers and improving design quality. Emmert-Streib, (2023) highlighted the role of LLMs and other AI techniques in digital twin systems, enabling real-time monitoring and predictive maintenance. Xia et al., (2024) demonstrated that orchestrating multiple LLM-based agents can automate simulation parameterisation, reduce user workload and support complex decision making. McClelland, (2022) argued that generative AI can significantly reduce product development time and costs, particularly in Computer-Aided Design (CAD) and Manufacturing (CAM) tools. While traditional computerised tools may require months or years to bring a product from concept to realisation, generative AI can compress this timeline to hours or days. This is particularly relevant for 3D manufacturing simulation software such as Visual Components, where simulation experts perform roles analogous to product developers in CAD/CAM environments. Additionally, Listl et al., (2023) illustrated the value of knowledge graph-based frameworks for automating and validating simulation models, emphasising the importance of structured knowledge representation.

2.6.1 LLMs and RAG in manufacturing simulation

Recent research highlights several applications of LLMs and RAG in manufacturing simulation:

Requirements-driven factory layout planning:

Tinsel et al., (2024) proposed using LLMs to automate the transformation of unstructured customer requirements into initial simulation model placeholders. LLMs identify necessary processes and machines, estimate dimensions, and generate simulation-ready outputs. Validation is enhanced by synthetic expert systems trained with LLMs, which learn probabilistic correlations between plant types, machines, and processes, improving both model generation and validation.

Multi-agent systems for digital twin parametrisation:

Xia et al., (2024) demonstrated that LLM-based multi-agent frameworks can automate complex parametrisation tasks in digital twin simulations. Specialised agents for observation, reasoning, decision-making, and summarisation enable dynamic interaction and reduce cognitive load for users.

Manufacturing equipment selection:

Werheid et al., (2024) investigated the use of LLMs combined with RAG to optimise equipment selection during ramp-up planning. Their factual-driven copilot integrates structured and semi-structured knowledge retrieval, providing guided, traceable recommendations. Industrial validation demonstrates the system's capability to provide logical and actionable recommendations for automation equipment, with evaluation showing that among 22 equipment prompts analysed, 19 involved selecting correct equipment while considering most requirements, and 6 cases fully met all requirements.

2.6.2 The role of ontologies in manufacturing simulation

Ontologies are essential for maximising the effectiveness of LLM and RAG systems in manufacturing simulation, as they provide formal vocabularies and explicit relationships between domain concepts. This structure enables accurate information retrieval, semantic interoperability, and context-aware reasoning. In LLM-augmented simulation, ontologies help interpret complex queries, integrate heterogeneous data sources, and generate responses grounded in simulation semantics. For example, when selecting robot models for a palletising task, ontologies ensure retrieved options are compatible with task requirements.(Sapel et al., 2024).

Recent applied research demonstrates the synergy between ontologies and RAG systems in engineering applications. For example, Yang et al., (2025) developed an ontology-driven framework for automotive assembly process knowledge exploration, where LLMs interpret user queries, perform semantic search, and provide answers grounded in a domain-specific ontology. Their results show that integrating LLMs with ontological knowledge structures significantly improves the accuracy and relevance of information

retrieval and reasoning in complex engineering domains, highlighting the value of ontologies as enablers for LLM-powered simulation and knowledge management systems.

Ontologies in manufacturing simulation are structured in a four-tier hierarchy (Figure 14), ranging from abstract, domain-independent concepts to specialised, application-specific implementations(Sapel et al., 2024).

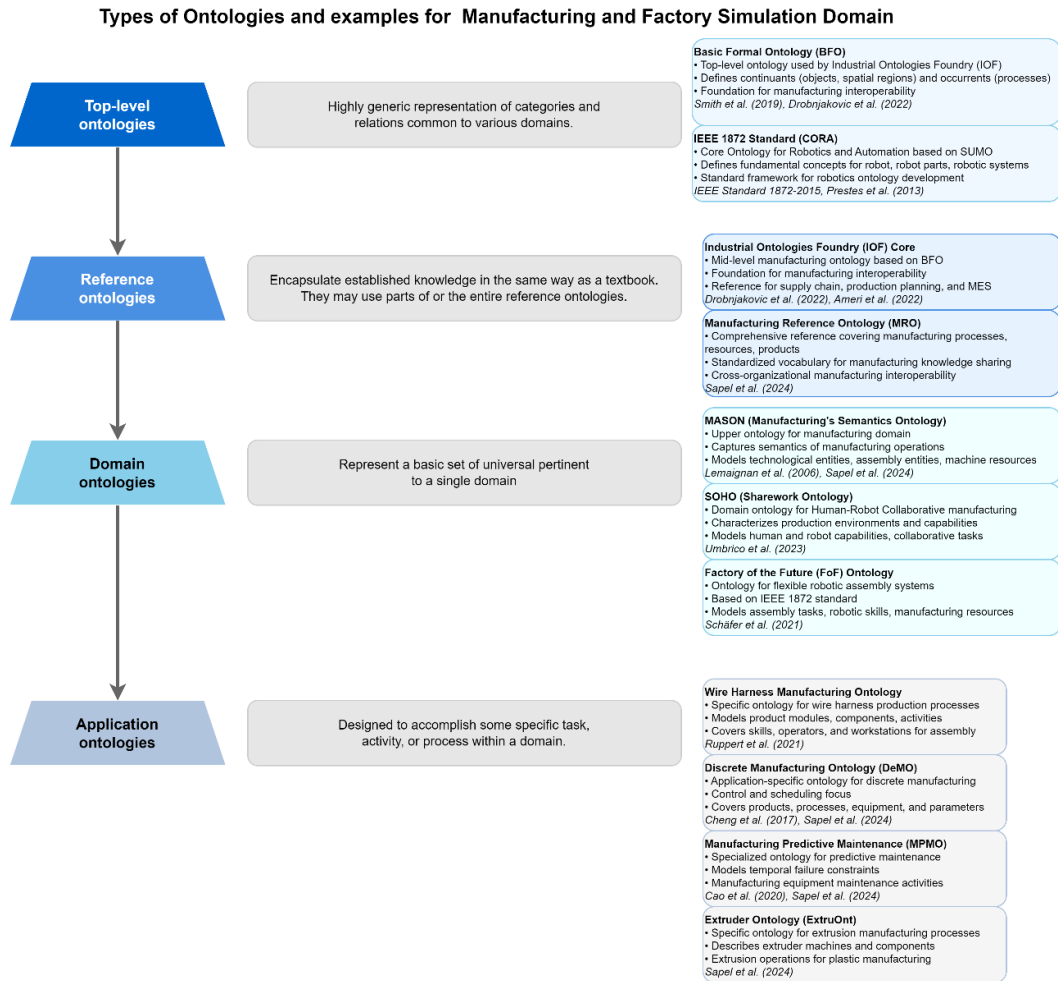


Figure 14. Types of ontologies and examples for manufacturing and factory simulation domain (Sapel et al., 2024).

Top-level ontologies-Foundational abstractions:

Top-level ontologies provide universal categories for simulation entities and events, such as 'process', 'resource', and 'event'. In manufacturing simulation, these ontologies establish the conceptual backbone for modelling scenarios, ensuring consistency across different simulation tools and data sources. The Basic Formal Ontology (BFO)(Arp et al., 2015) and the IEEE 1872 Standard based Core Ontology for Robotics and Automation (CORA)(IEEE, 2015; Prestes et al., 2013) are prominent examples, offering

foundational frameworks for comprehensive manufacturing system modelling.

Reference ontologies-Knowledge repositories:

Reference ontologies serve as comprehensive knowledge repositories, analogous to textbooks in their scope and authority. In simulation, they facilitate the integration of simulation data with enterprise systems (e.g., MES, ERP), supporting end-to-end digital threads and traceability in simulation workflows. The Industrial Ontologies Foundry (IOF) Core ontology (Drobnjakovic et al., 2022) and the Manufacturing Reference Ontology (MRO) (Usman et al., 2013) exemplify this level, providing standardised vocabularies and frameworks for supply chain management, production planning, and manufacturing execution across diverse organisations.

Domain ontologies-Specialised manufacturing frameworks:

Domain ontologies capture essential concepts and relationships specific to manufacturing while maintaining broad applicability. In simulation, they model entities such as robots, conveyors, human operators, and collaborative tasks, enabling more realistic and interoperable simulation models. Examples include MASON (Manufacturing's Semantics Ontology) (Lemaignan et al., 2006), which models technological entities, assembly entities, machine resources, and manufacturing operations; SOHO (Sharework Ontology for Human-Robot Collaboration), which addresses collaborative manufacturing (Umbrico et al., 2020); and the Factory of the Future (FoF) Ontology (Schäfer et al., 2021), which supports autonomous manufacturing systems.

Application ontologies-Task-specific implementations:

Application ontologies are designed for specific tasks, activities, or processes within manufacturing domains. In simulation platforms, they encode task-specific semantics for specialised processes (e.g., wire harness assembly, predictive maintenance), allowing simulation engineers to configure, validate, and optimise scenarios with domain-specific semantics. Examples include the Wire Harness Manufacturing Ontology (Nagy et al., 2021), Discrete Manufacturing Ontology (DeMO) (Cheng et al., 2017), Manufacturing Predictive Maintenance Ontology (MPMO) (Cao et al., 2020), and the Extruder Ontology (ExtruOnt) (Ramírez-Durán et al., 2020), each addressing specialised manufacturing processes while maintaining ontological rigour.

3 Research material and methods

This chapter describes the research design and methodology used in the thesis. It details the data sources, preparation processes, and experimental frameworks developed to evaluate the impact of data structures on RAG performance in manufacturing simulation. The chapter also introduces the question-answer (QA) dataset and the setup for the three main case studies.

3.1 Research design

This thesis employs a comparative experimental design to assess three RAG systems, vector store-based RAG, GraphRAG, and SQL agentic RAG, using proprietary data from Visual Components. Each system represents a distinct data architecture: vector stores, knowledge graphs, and relational databases. The experimental flow for each case is illustrated in Figure 15.

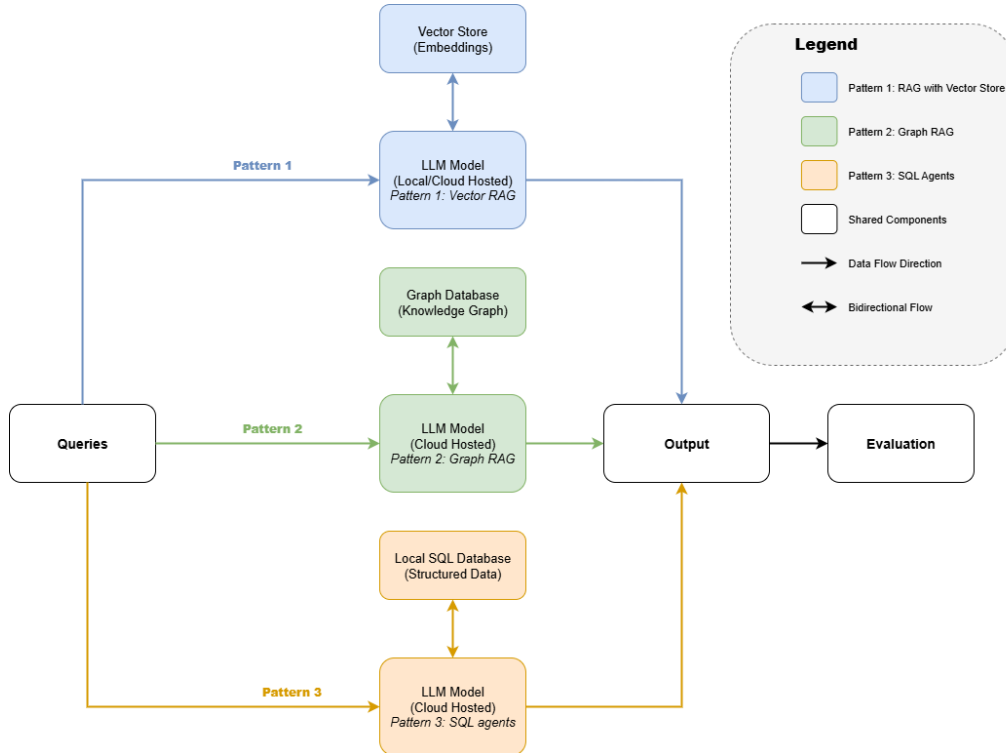


Figure 15 Overview of three case studies generic flow.

The Figure 15 shows three experimental patterns evaluated in this thesis. Pattern 1 Vector RAG: a retriever over a vector store supplies context to an LLM to produce an answer. Pattern 2 Graph RAG: the query is routed to a Neo4j knowledge graph; the LLM consumes context from Neo4j database to generate an answer. Pattern 3 SQL Agents: an LLM agent interacts with an SQLite database to pose queries and return answers. Outputs from all

patterns feed a common evaluation approach involving automated and human evaluation.

Experiments are conducted using pre-trained LLMs without fine-tuning, and all systems are tested with consistent query sets, embedding models, and evaluation protocols.

3.2 Data sources and preparation

This study uses a multi-layered data preparation strategy to evaluate RAG systems in manufacturing simulation. The primary data source is the eCatalog metadata of 3,452 predesigned components from Visual Components, stored in relational format and transformed into four representations to support different retrieval architectures:

JSON: Structured data

RDF: Resource description format

SQLite3: Relational table with full-text and vector indexes

Neo4j: Knowledge graph with full-text and vector indexes.

To develop the experimental query dataset, a Microsoft Form survey was conducted among domain experts, collecting natural language queries typical for Visual Components software. These queries were categorised according to the theoretical framework (see section 2.5.3 and demonstrated with examples below.

Explicit Fact Queries:

- 1) How many components are available?
- 2) What new components have been added after the latest release?
- 3) What are the most important components?

These queries seek direct information from the dataset, such as counts, updates, or lists of key components.

Implicit Fact Queries:

- 1) Recommend collaborative robot models with a 5kg payload and at least 1 meter reach for assembly operations.
- 2) Which AGVs have a payload of at least 100 kg and are the fastest?
- 3) Which AGV models have the longest battery endurance? How long can they operate without recharging? How far can they travel without recharging?

These queries require logical inference or combining multiple criteria, such as filtering models by payload, speed, or endurance. The system must synthesise information across different data fields to provide relevant recommendations.

Interpretable Rationale Queries:

- 1) What components are needed to build a machine tending/intralogistics/palletizing layout?
- 2) Find a welding/palletizing/material handling robot with a specific reach and payload.

- 3) I want to model a visualization layout for an electronics manufacturing concept including AGVs, robot arms, hu-man workstations, and automated storage systems. Recommend a set of 20 components to use.

These queries demand domain-specific reasoning, requiring the system to understand application requirements and recommend suitable components or configurations based on context and user objectives.

Hidden Rationale Queries:

- 1) Why don't I see the latest models in Explorer?
- 2) Why is the panel blank? What's wrong?
- 3) How to model a specific type of component?

These queries involve diagnostic or contextual understanding, such as troubleshooting visibility issues or guiding users through modelling processes. The system must interpret underlying causes and provide rationale or step-by-step guidance.

These real-world queries informed the development of the QA dataset used for benchmarking RAG system performance.

3.2.1 QA dataset generation pipeline

To ensure consistency and fairness across all retrieval experiments, a unified and high-quality question and answer (QA) dataset was developed using an automated, data-driven pipeline (see Figure 16). The primary goal was to eliminate human bias and generate diverse queries reflecting both factual and inferential information needs.

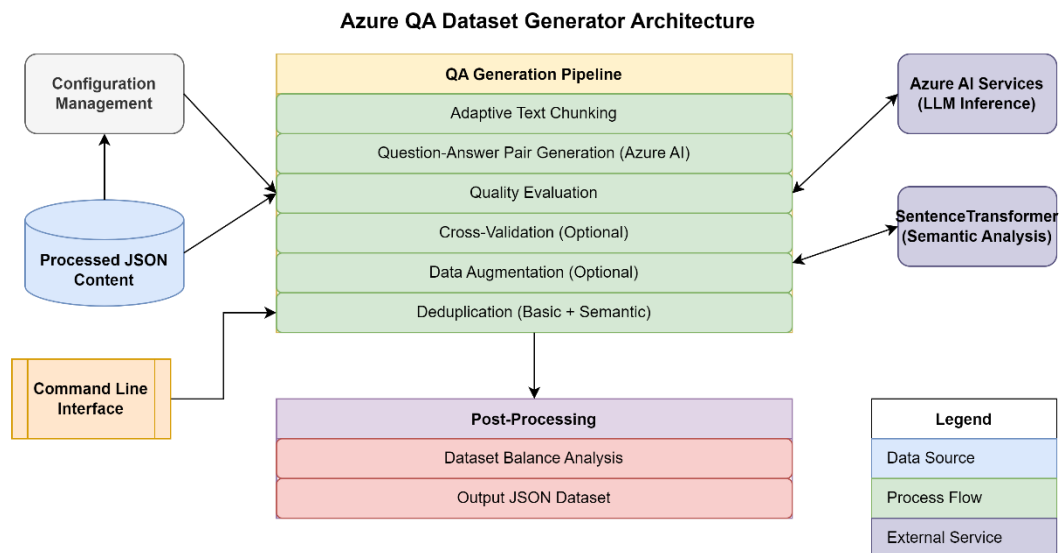


Figure 16 Implicit and explicit QA dataset generator.

The pipeline begins by converting the core item table from SQL to structured JSON, which is then processed by Azure AI services. Adaptive chunking segments the data along semantic boundaries (e.g., paragraphs) to preserve

context. The pipeline generates QA pairs using gpt-4o, applies prompt engineering for a balanced mix of query types, and evaluates quality through both heuristic scoring and optional LLM-based cross-validation. Data augmentation (optional) introduces alternative phrasings, and deduplication (basic and semantic) ensures uniqueness. Post-processing includes dataset balance analysis and output formatting.

To address quality, QA pairs were penalised for short or generic answers and hallucinated content, with only those above a set threshold retained. Semantic deduplication used a Sentence Transformer model to remove conceptually duplicate questions, keeping the highest-quality variant.

The final dataset comprised 4,140 QA pairs (3,260 explicit, 880 implicit), with an average question length of 10.8 words and answer length of 16.3 words. Figure 17 shows a sample of the dataset structure, illustrating both explicit and implicit queries.

```
{
  "explicit_queries": [
    {
      "query": "What tool is used to move the robot in the programming tab?",
      "ground_truth": "The 'Jog' tool is used to move the robot in the programming tab."
    },
    {
      "query": "What is the purpose of the Quality check camera?",
      "ground_truth": "The Quality check camera can be used as decoration or as a template where custom behaviors can be added."
    }
  ],
  "implicit_queries": [
    {
      "query": "Why might the 'Robot Transport Controller' component be useful in process modeling?",
      "ground_truth": "The 'Robot Transport Controller' component can be used to automate robot programming, making process modeling more efficient."
    },
    {
      "query": "What can be inferred about the functionality of the AGV Overview layout?",
      "ground_truth": "The AGV Overview layout likely provides guidance and tools for effectively setting up and managing AGVs, including ensuring safety and optimizing pathways."
    }
  ]
}
```

Figure 17 Sample and structure of QA pairs.

The dataset was manually reviewed for clarity and relevance, resulting in 2,167 accepted pairs out of 4,053 reviewed. For benchmarking, subsets of 25, 50, and 100 QA pairs per category were created as separate json files for each subset. These json structured QA dataset files serve as a foundational resource for evaluating the three RAG system case studies described in section 3.4.

3.3 Common technological stack used

Table 1 summarises the core technology stack used in this thesis. Python is the main programming language, managed in a virtual environment within

Visual Studio Code. LangChain serves as the primary orchestration framework, supporting both cloud-based models via Azure AI Foundry (Microsoft, 2025f) and local models via Ollama for windows(Meta Ollama, 2025). Additional libraries for data processing and database management are listed in the table below.

Table 1 Common technology stack across experimentation cases.

Category	Technologies
Programming language and environment	Python 3.x - Core programming language python-dotenv - Environment variable management Visual Studio Code Virtual Environment - Isolated dependency management
AI/LLM frameworks and python libraries	LangChain: Packages: langchain-core, langchain-community, langchain-ollama, langchain-azure-ai.
Cloud models provider	Azure AI Foundry
Local model access	Ollama for windows
Data processing libraries	NumPy, Pandas, JSON, scikit-learn, openpyxl
RDF python libraries	RDFLib
SQLite python libraries	SQLite, sqlite-vec, SQL alchemy
Neo4j python libraries	Neo4j for Neo4j driver Neo4j-graphrag for GraphRAG framework
Other desktop applications	Neo4j Desktop, Microsoft SQL Server Management Studio

For prompt engineering, a consistent data description was used, detailing each variable's contextual meaning as shown in Table 2.

Table 2 Data field and their contextual definition.

Field Name	Definition
Name	The display name of the component or layout.
VCID	A unique identifier for the component or layout.
Modified	The timestamp of the last modification.
ModelPopularity	A numeric value indicating the popularity or usage frequency of the model.
Description	A textual description of the component, its use, or key features.
ModelType	The type of model, e.g., 'Component' or 'Layout'.
Type	The category or type of the component or layout (e.g., Robots, Conveyors, Machines, etc.).
Manufacturer	The company or organization that manufactures the component.
Author	The author or creator of the component or layout.
Revision	The revision or version number of the component or layout.
MaxPayload	The maximum payload capacity of the component, if applicable (in kg).
FileUri	A URI or link to the file or resource representing the component or layout.
Subfolder	The hierarchical folder or path where the component or layout is stored.
Tags	A semicolon- or comma-separated list of tags or keywords describing the component.

Website	A URL to the manufacturer's or component's website.
Email	A contact email address for the manufacturer or author.

Various LLM and embedding models were used, both locally and in the cloud, with key parameters summarised in Table 3. Local model selection was limited by laptop hardware, while cloud model selection depended on available top models at the time of experimentation.

Table 3 Used LLMs and embedding models with basic info.

Hosted Environment	Model Type	Model Name	Architecture	Parameters	Context Length	Embedding Length	Quantization	Capabilities
Local	LLM Model	llama3	llama	8.0B	8192	4096	Q4_0	Completion
Local	Embedding Model	bge-m3	bert	566.70M	8192	1024	F16	Embedding
Cloud	LLM Model	gpt-4o	GPT	16.4B	16,384	4096	-	Multimodal (text, image, audio)(Microsoft, 2025a)
Cloud	Embedding Model	text-embedding-ada-002	GPT-3	1.5B	8192	1536	-	Text search, code search, sentence similarity(Microsoft, 2025b)
Cloud	Embedding Model	text-embedding-3-large	GPT-3	1.5B	8192	3072	-	Text search, code search, sentence similarity(Microsoft, 2025c)

3.4 Experimentation frameworks

This section describes the specific architectures and workflows for FAISS vector store-based RAG, GraphRAG with Neo4j, and agentic RAG with relational databases, detailing how each approach was implemented and assessed. These frameworks are not merely theoretical; they are directly reflected in the actual python programs developed and executed as part of this thesis.

3.4.1 FAISS vector store-based RAG

This section presents the experimental framework designed to assess the performance of RAG systems utilising FAISS vector stores, with data corpora formatted as either JSON or RDF.

1) FAISS vector store-based RAG(JSON)

The FAISS-based RAG system with a json corpus employs two coordinated pipelines: one transforms raw product data into dense and sparse retrieval

structures, while the other manages end-to-end query processing, including context retrieval, answer generation, and evaluation.

Pipeline 1: From raw json corpus to embedded hybrid store

As illustrated in Figure 18, the initial pipeline transforms structured product metadata from a curated json corpus into a hybrid retrieval system that integrates both semantic and lexical search capabilities. Each json object is converted into a LangChain Document abstraction, encapsulating text and metadata to ensure schema independence.

A central configuration module manages all experimental parameters, including model provider selection (Azure-hosted embeddings or local Ollama backend), file paths, index settings, and ensemble weights. This modular approach facilitates reproducibility and enables seamless switching between local and cloud-based inference.

Two retrieval backends are established:

- Dense Vector Index (FAISS): Documents are embedded using the selected model and stored in a FAISS index, supporting semantic similarity search.
- Sparse Lexical Index (BM25): Token distributions are indexed via BM25, optimising keyword-based retrieval for explicit queries.

To address rate limits and cost constraints associated with embedding services, a custom rate-limiting wrapper is applied, ensuring stable throughput during large-scale ingestion and re-indexing. Finally, the system fuses dense and sparse retrieval modes through a weighted linear combination of scores, enabling fine-tuning between lexical precision and semantic recall. Comprehensive logging at each stage supports performance tuning and reproducibility.

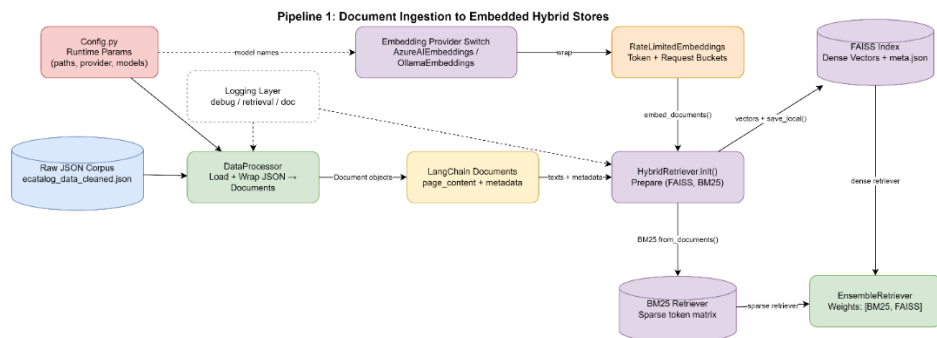


Figure 18 Illustrates the transformation of the raw data corpus into the hybrid vector store.

Pipeline 2: From user query to answer, evaluation, and optimisation

Figure 19 depicts the second pipeline, which processes user queries and benchmark test sets. Queries are optionally normalised or tagged to distinguish explicit from implicit intent, supporting stratified evaluation.

The hybrid retriever fetches candidate documents from both FAISS and BM25 in parallel, fusing their scores according to configured weights. An

optional Maximal Marginal Relevance (MMR) step may be applied to minimise redundancy and maximise contextual diversity in retrieved documents.

The top K documents are assembled into a prompt template containing explicit system instructions, directing the language model to cite only the provided context or acknowledge insufficiency, thereby mitigating hallucination risk and ensuring grounded generation. The generation component dynamically selects between cloud-hosted (e.g., gpt-4o via Azure Foundry) and locally hosted (e.g., Llama 3 via Ollama) models, with rate-limiting mechanisms in place to respect resource constraints.

Following generation, the evaluation module assesses response quality using key metrics:

- Semantic similarity: Calculated between the generated answer and ground-truth reference using the same embedding backbone.
- Retrieval quality: Measured via precision, recall, and F1 score over relevant documents.
- Success flag: A binary indicator based on a cosine similarity threshold, used for aggregated success-rate reporting.

Checkpointing is implemented to support fault-tolerant benchmarking, with results exported in structured formats (CSV, Excel) for subsequent statistical analysis and visualisation.

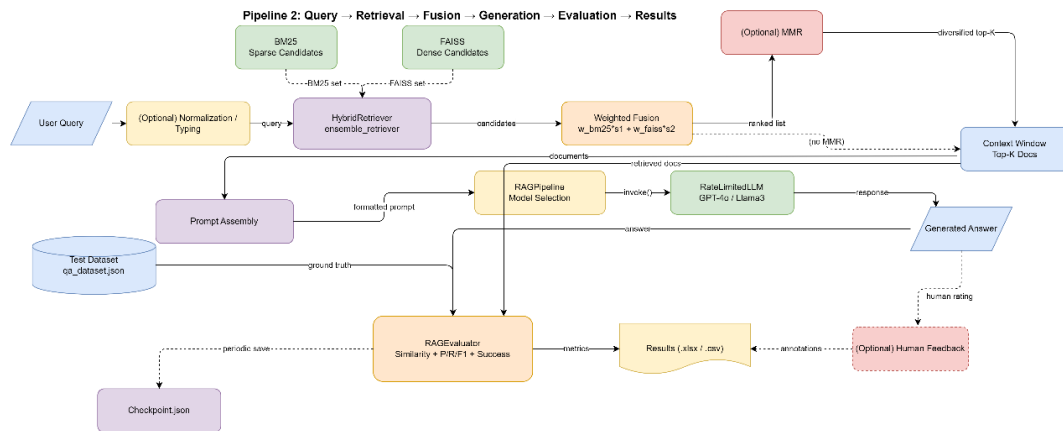


Figure 19 Query to evaluation pipeline.

Practical implementation aspect:

The practical implementation of the FAISS Vector Store based RAG system involves several key steps, from data loading and preprocessing to hybrid retrieval, prompt construction, and evaluation. Below are the main code snippets and their explanations focusing on evaluation pipeline.

a) Data loading and preprocessing:

The evaluation employs a structured JSON dataset with two query categories: explicit and implicit queries, each paired with ground truth answers as shown earlier in Figure 17 and described in section 3.2.1. The dataset file is loaded first. The preprocessing function (Figure 20) extracts both query

collections, annotates each instance, and consolidates them into a unified corpus.

```
def load_test_data(path: str):
    with open(path, "r", encoding="utf-8") as f:
        data = json.load(f)
    explicit = data.get("explicit_queries", [])
    implicit = data.get("implicit_queries", [])
    for x in explicit:
        x["query_type"] = "explicit"
    for x in implicit:
        x["query_type"] = "implicit"
    return explicit + implicit
```

Figure 20 Data loading implementation python code snippet.

b) Hybrid retriever initialization:

Before query processing begins, the system constructs two retrieval indices from the document corpus. The hybrid retrieval mechanism combines BM25 lexical retrieval with FAISS-based dense vector retrieval. A BM25 inverted index captures keyword statistics, while a FAISS vector index encodes semantic representations. These operate in parallel and are unified through an ensemble retriever with configurable weights (e.g., BM25: 0.3, FAISS: 0.7). The class shown in Figure 21 sets up the hybrid retrieval system, allowing for flexible weighting between semantic and lexical search.

```
class HybridRetriever:
    def initialize_stores(self, documents):
        self.faiss_store = FAISS.from_texts(texts, self.embeddings)
        self.bm25_retriever = BM25Retriever.from_texts(texts)
        self.ensemble_retriever = EnsembleRetriever(
            retrievers=[self.bm25_retriever, self.faiss_retriever],
            weights=config.ENSEMBLE_WEIGHTS # [0.3, 0.7]
        )
```

Figure 21 Hybrid retrieval implementation python code snippet.

c) Query processing and response generation:

For each query, the system executes a multi-stage pipeline:

Stage 1: Query Encoding: The input query undergoes dual encoding: (1) converted to a dense embedding vector for FAISS semantic search, and (2) tokenized into keywords for BM25 lexical matching. The ensemble retriever processes both representations simultaneously shown in Figure 22.

Stage 2: Document Retrieval: The ensemble retriever ranks candidate documents by combining BM25 keyword-based scores and FAISS semantic similarity scores with configurable weights.


```

def get_relevant_documents(self, query: str):
    # BM25: tokenize query into keywords, match against inverted index
    bm25_docs = self.bm25_retriever.get_relevant_documents(query)

    # FAISS: embed query into vector, compute cosine similarity with doc embeddings
    query_embedding = self.embeddings.embed_query(query)
    faiss_docs = self.faiss_store.similarity_search_by_vector(query_embedding, k=FAISS_K)

    # Ensemble: combine with weights [BM25_weight, FAISS_weight]
    combined_docs = self.ensemble_retriever.get_relevant_documents(query)
    return combined_docs

```

Figure 22 Ensemble retrieval process python code snippet.

Stage 3: Context window management: Retrieved documents are accumulated incrementally (Figure 23) while respecting the model's token limits.

```

def _accumulate_docs_with_limit(self, docs, question, prompt_template):
    context_window = config.MODEL_CONTEXT_WINDOW # e.g., 120,000
    reserve_completion = config.RESERVED_COMPLETION_TOKENS # e.g., 4,000
    margin = config.SAFETY_MARGIN_TOKENS # e.g., 1,000
    hard_cap = context_window - reserve_completion - margin

    included = []
    running_context = []
    for doc in docs:
        tentative_context = "\n\n".join(running_context + [doc.page_content])
        prompt_candidate = prompt_template.format(context=tentative_context, question=question)
        tokens = self._estimate_tokens(prompt_candidate)
        if tokens <= hard_cap:
            running_context.append(doc.page_content)
            included.append(doc)
        else:
            break # Stop adding documents

    final_context = "\n\n".join(running_context)
    prompt = prompt_template.format(context=final_context, question=question)
    return {'prompt': prompt, 'included_docs': included, 'prompt_tokens': tokens}

```

Figure 23 Document accumulation with token limits python code snippet.

Stage 4: LLM generation: The formatted prompt (query + context) is sent to the language model to generate an answer. The function in Figure 24 retrieves relevant documents, constructs the prompt, and invokes the LLM to generate an answer.

```

def run(question: str):
    raw_docs = retriever.get_relevant_documents(question)
    build = self._accumulate_docs_with_limit(raw_docs, question, prompt_template)
    prompt_str = build['prompt']

    # Invoke LLM with constructed prompt
    response = self.llm.invoke(prompt_str)
    answer = response.content

    return {
        "answer": answer,
        "context": build['included_docs'],
        "prompt_token_count": build['prompt_tokens'],
        "completion_token_count": self._estimate_tokens(answer)
    }

```

Figure 24 LLM invocation python code snippet.

Stage 5: Post-processing: For evaluation purposes, retrieved documents are assessed for relevance and results are recorded. The loop in Figure 25 processes each test query, evaluates document relevance, and stores results for further analysis.

```

for item in test_items:
    response = pipeline(item["query"])
    retrieved_docs = response.get("context", [])
    for doc in retrieved_docs:
        relevant = cosine_relevance(doc.page_content, item["ground_truth"],
                                   embeddings, threshold=0.4)
        retrieved_docs.append({"content": doc.page_content, "relevant": relevant})
    results.append({
        "query": item["query"],
        "response": response.get("answer"),
        "ground_truth": item["ground_truth"],
        "retrieved_docs": retrieved_docs,
        "query_type": item.get("query_type")
    })

```

Figure 25 Post processing loop python code snippet.

d) Evaluation and results aggregation:

The evaluation process computes a comprehensive suite of metrics spanning retrieval effectiveness and generation quality. Retrieval performance is quantified through precision at k (proportion of retrieved documents deemed relevant), recall at k (proportion of relevant documents successfully retrieved), F1-score (harmonic mean of precision and recall), and Mean Reciprocal Rank (MRR, measuring the rank position of the first relevant document). Generation quality is assessed via three complementary metrics: cosine similarity between the generated response and ground truth answer (with a threshold of 0.8 defining binary success), context relevancy (semantic alignment between query and retrieved documents), and faithfulness (degree to which the generated answer is grounded in the retrieved context). The implementation shown in Figure 26 computes these metrics for each query instance.

```

class RAGEvaluator:
    def evaluate(self, results):
        for r in results:
            docs = r.get("retrieved_docs", [])
            precision = len([d for d in docs if d["relevant"]]) / len(docs)
            recall = len([d for d in docs if d["relevant"]]) / total_relevant
            f1_score = 2 * precision * recall / (precision + recall)
            mrr = 1.0 / (position + 1) # position of first relevant doc
            cosine_truth = semantic_similarity(r["response"], r["ground_truth"])
            success = cosine_truth >= 0.8
            context_relevancy = avg_cosine(query, retrieved_docs)
            faithfulness = max_cosine(answer, retrieved_docs)

```

Figure 26 Evaluation metrics python code snippet.

Results are aggregated (Figure 27) by computing summary statistics for all queries, explicit queries, and implicit queries separately. Results are exported to Excel with multiple sheets: per-query results, summary metrics (success rate, precision, recall, F1, MRR), success distribution by query type, and run configuration (model, ensemble weights, retrieval parameters).

```

summary_stats = {
    "All Queries": compute_metrics(df),
    "Explicit Queries": compute_metrics(df[df.query_type == "explicit"]),
    "Implicit Queries": compute_metrics(df[df.query_type == "implicit"])
}
# Export: Per_Query, Summary, Success_Pivot, Run_Config sheets

```

Figure 27 Result aggregation python code snippet.

e) Human feedback integration:

The system incorporates human feedback annotation in two modes: immediate feedback during evaluation and post-evaluation batch annotation. Immediate mode(Figure 28) prompts real-time correctness assessment during query processing. In this thesis, immediate feedback annotation is used. Annotations are stored in “Human_Feedback” and “Human_Feedback_Metrics” sheets of excel workbook.

```
if config.IMMEDIATE_HUMAN_FEEDBACK:
    print("\nQuery:", item["query"])
    print("Ground Truth:", item["ground truth"])
    print("Response:", resp.get("answer", ""))
    fb = input("Is the response correct? (True/False blank=skip): ").strip().lower()
    if fb in ("true", "false"):
        human_feedback = (fb == "true")
    results.append({
        # ... automated metrics ...
        "human_feedback": human_feedback
    })
```

Figure 28 Immediate human feedback implementation.

The final excel workbook created in the evaluation process is illustrated in Figure 29.

query	response	found	truth	type	decision	at	recall	at	fi	score	mrr	hntic	simil	ext	relev	faithfulness	ne	with	ti	success	lm	retrieval	relevance	response	tript	token	lion	token	flow	truncated	dncated	chhan	feedback
What type Robot Wo The R2L-St explicit		1	1	1	1	1	1	1	1	1	1	0,747836	0,774747	0,800332	0,909336	TRUE	10	10	0,991407	3450	5	3455	FALSE	0	0	TRUE							
What is th The takt ti The takt ti explicit		1	1	1	1	1	1	1	1	1	1	0,926232	0,756945	0,807676	1	TRUE	10	10	0,730358	3585	12	3597	FALSE	0	0	TRUE							
What is th The primal The easym explicit		1	1	1	1	1	1	1	1	1	1	0,957991	0,793859	0,863454	0,975563	TRUE	10	10	0,699362	3222	23	3245	FALSE	0	0	TRUE							
What com The Bulk Fi The Bulk Fi explicit		1	1	1	1	1	1	1	1	1	1	0,924849	0,8077	0,888098	0,967693	TRUE	10	10	0,810988	3068	39	3107	FALSE	0	0	TRUE							
What does The Towle The Towle explicit		1	1	1	1	1	1	1	1	1	1	0,913578	0,780245	0,831992	0,972472	TRUE	10	10	0,731954	3333	32	3365	FALSE	0	0	TRUE							
What is th 6" discs. The Electri explicit		1	1	1	1	1	1	1	1	1	1	0,859741	0,758277	0,774605	0,856744	TRUE	10	10	0,777364	2907	4	2911	FALSE	0	0	TRUE							
Who is the Visual Con The manu explicit		1	1	1	1	1	1	1	1	1	1	0,726576	0,76493	0,793004	0,836454	TRUE	10	10	0,53244	2425	2	2427	FALSE	0	0	TRUE							
What is th Single Axis The comp explicit		1	1	1	1	1	1	1	1	1	1	0,762458	0,798302	0,825661	0,888886	TRUE	10	10	0,65753	3130	6	3136	FALSE	0	0	TRUE							
Who is the Comau The manu explicit		1	1	1	1	1	1	1	1	1	1	0,765668	0,758818	0,786537	0,856388	TRUE	10	10	0,565794	2447	2	2449	FALSE	0	0	TRUE							
What hasg When a ro When a ro explicit		1	1	1	1	1	1	1	1	1	1	0,952341	0,808935	0,868569	1	TRUE	10	10	0,958499	3289	41	3330	FALSE	0	0	TRUE							
Who is the Visual Con The manu explicit		1	1	1	1	1	1	1	1	1	1	0,713108	0,796586	0,790473	0,810251	TRUE	10	10	0,807873	2816	2	2818	FALSE	0	0	TRUE							
Who is the JOT Autom The manu explicit		1	1	1	1	1	1	1	1	1	1	0,739969	0,773026	0,777516	0,885267	TRUE	10	10	0,642694	2386	3	2389	FALSE	0	0	TRUE							
What type Robot Wo The B2C-2 explicit		1	1	1	1	1	1	1	1	1	1	0,740631	0,778265	0,80266	0,912181	TRUE	10	10	0,657853	3326	5	3331	FALSE	0	0	TRUE							
What type I don't hav The PT101 explicit		1	1	1	1	1	1	1	1	1	1	0,720557	0,755982	0,715117	0,728514	FALSE	10	10	0,695626	3773	6	3779	FALSE	0	0	FALSE							
What is th The primal The primar explicit		1	1	1	1	1	1	1	1	1	1	0,94629	0,762037	0,857873	1	TRUE	10	10	0,757311	3560	20	3580	FALSE	0	0	TRUE							
What is th Visual Con The manu explicit		1	1	1	1	1	1	1	1	1	1	0,780828	0,790408	0,81074	0,854669	TRUE	10	10	0,631184	2340	2	2342	FALSE	0	0	TRUE							
What is th Componer The model explicit		1	1	1	1	1	1	1	1	1	1	0,754981	0,769154	0,784249	0,818716	TRUE	10	10	0,541009	2863	1	2864	FALSE	0	0	TRUE							
Who is the Reis The manu explicit		1	1	1	1	1	1	1	1	1	1	0,738431	0,795302	0,747251	0,828758	TRUE	10	10	0,538013	2546	2	2548	FALSE	0	0	TRUE							
What is th Single Axis The type o explicit		1	1	1	1	1	1	1	1	1	1	0,762023	0,776077	0,825661	0,890316	TRUE	10	10	0,618944	3410	6	3416	FALSE	0	0	TRUE							
What are I I don't hav The dimen explicit		1	1	1	1	1	1	1	1	1	1	0,724542	0,812062	0,717091	0,715031	FALSE	10	10	0,767941	3254	6	3260	FALSE	0	0	FALSE							
Who is the Visual Con The manu explicit		1	1	1	1	1	1	1	1	1	1	0,735255	0,801058	0,793196	0,817553	TRUE	10	10	0,570863	2314	2	2316	FALSE	0	0	TRUE							
What is th CIMTEC Ai The manu explicit		1	1	1	1	1	1	1	1	1	1	0,742293	0,764971	0,791838	0,872376	TRUE	10	10	0,603308	2999	4	3003	FALSE	0	0	TRUE							
What mus I don't hav To enable explicit		1	1	1	1	1	1	1	1	1	1	0,722166	0,760323	0,716173	0,705148	FALSE	10	10	0,4744533	3855	6	3861	FALSE	0	0	FALSE							
What is th Placing bo The primar explicit		1	1	1	1	1	1	1	1	1	1	0,813956	0,778018	0,807714	0,898434	TRUE	10	10	0,680618	3127	13	3140	FALSE	0	0	TRUE							

Figure 29 Sample of final excel workbook with multiple sheets.

These code snippets collectively demonstrate the end-to-end workflow for evaluating a FAISS Vector Store based RAG system, from data loading and hybrid retrieval to prompt construction, LLM invocation, and both automated and human evaluation. For the remainder of the experimental frameworks, code snippets are not presented, as the overall process from query to response and final Excel result generation remains consistent, differing only in the underlying data transformation and storage approaches and their respective retrieval and generation pipelines. In cases where code cannot be disclosed, this is due to confidentiality considerations.

2) FAISS vector store-based RAG(RDF)

Building on the json-based framework, the RDF approach introduces a semantically enriched structure aligned with ontology-driven principles. While json offers flexibility, RDF formalises relationships between entities. The

modular ETL pipeline (Figure 30) converts json eCatalog metadata into a standards-compliant RDF graph using a fixed ontology, with optional subclass extensions. The output is exported in multiple serialisation formats.

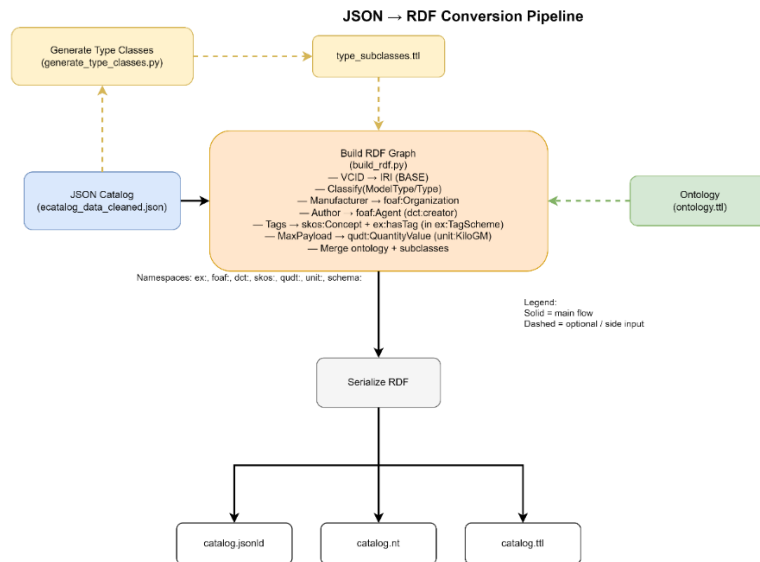


Figure 30 JSON to RDF data ETL pipeline.

BM25 and FAISS indexes are then constructed from the serialised RDF data, supporting both structured and Turtle formats for indexing. The overall RAG process, shown in Figure 31, mirrors the JSON-based pipeline but leverages RDF as the initial input.

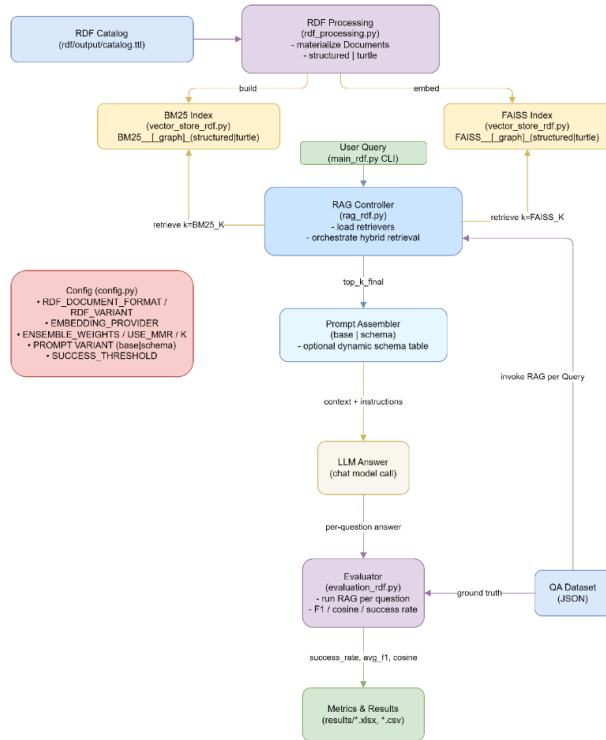


Figure 31 Overall architecture of FAISS vector store-based RAG starting data corpus as RDF serialized file.

For embedding generation, documents are processed in two formats:
 Structured format: Each eCatalog entry is rendered as a flat, structured document (Figure 32).
 Turtle format: Each entry is represented in pure Turtle syntax (Figure 33).

```

-----BEGIN DOCUMENT (STRUCTURED | FLAT)-----
== Asset ==
URI: http://example.org/robot-catalog/asset/f9613429-6187-415f-a85b-368868eb5d5
Name: WM 25-2_3_OFF
Description: The robots can be programmed in the programming tab.
In the case of conceptual layouts, the robots can utilize robot controllers to automatically perform tasks.
All the robots have a similar component structure and therefore they mostly support same features and they are easily replaceable with each other.
=== How to use ===
- Program tab: Move the robot with the "dog" tool and create the motion statements in the "Program Editor".
The created statements will have configurable properties as well for grasping/releasing products, changing tools, and other actions., see the "Signal Actions" and "Actions Configurations" properties for further customization.
=== How to use - Process Modeling ===
In process modeling, "Robot Transport Controller" component can be used to automate robot programming.
See our academy lessons for further instructions.
=== How to use - OLP ===
With Robotics OLP product you can program robot and post-process the job file to native robot language.
=== Key Properties ===
##Executor::IsEnabled - Determines if the robot is turned ON / OFF
##Executor::IsLooping - Determines if the robot program is looped when the program is run in programming tab
Manufacturer: Organization:Comau
Tags: Articulated, Small
Types: Industrial Asset, Robot
ClassHierarchy: Industrial Asset | Robot > Component > Industrial Asset
Payload: 25.0 unit:KiloGM
VCID: f9613429-6187-415f-a85b-368868eb5d5
CategoryType: Robots
ModelType: Component
Revision: 163
PopularityScore: 0.0
IsDeprecated: false
Modified: 2024-02-07T14:26:24
StoragePath: /Components/Comau/Robots/
FileURI: https://ecat.visualcomponents.net/e11b/4.9/eCat/Components/Comau/Robots/WM 25-2_3_OFF.vcmx
MaxPayload: 25.0
AuthorName: Visual Components
TagsLiteral: Articulated;Small
== AllProperties ==
creator: Visual_Components
== ClassDefinitions ==
Industrial Asset: An item in the industrial simulation catalog (component or layout). | Robot: Industrial or collaborative robot. | Component: A physical or logical component (robot, machine, actuator, etc.).
== PropertyDefinitions ==
creator
-----END DOCUMENT (STRUCTURED | FLAT)-----
  
```

Figure 32 Structured flat document format.

```

ex:revision a rdf:Property ;
  rdfs:label "revision" ;
  rdfs:comment "Revision / version number of the asset definition." ;
  rdfs:domain ex:IndustrialAsset ;
  rdfs:range xsd:integer .

ex:storagePath a rdf:Property ;
  rdfs:label "storage path" ;
  rdfs:comment "Hierarchical folder (subfolder) path in the source catalog." ;
  rdfs:domain ex:IndustrialAsset ;
  rdfs:range xsd:string .

ex:tags a rdf:Property ;
  rdfs:label "tags literal" ;
  rdfs:comment "Original semicolon or comma separated tag list as a single literal." ;
  rdfs:domain ex:IndustrialAsset ;
  rdfs:range xsd:string .

ex:vcId a rdf:Property ;
  rdfs:label "VCID" ;
  rdfs:comment "Unique Identifier of the component or layout (catalog primary key).";
  rdfs:domain ex:IndustrialAsset ;
  rdfs:range xsd:string .

<http://example.org/robot-catalog/asset/f9613429-6187-415f-a85b-360860e0d5d5> a ex:IndustrialAsset,
  ex:Robot ;
  ex:authorName "Visual Components" ;
  ex:categoryType "Robots" ;
  ex:description "The robots can be programmed in the programming tab. In the case of conceptual layouts, the robots can utilize robot controllers to automatically perform tasks.
  All the robots have a similar component structure and therefore they mostly support same features and they are easily replaceable with each other.
  ## How to use - Program tab ## How to use the robot with the "log" tool and create the action statements in the "Program Editor".
  The created statements will have configurable properties as well. For grasping/releasing products, changing tools, and other actions., see the "Signal Actions" and "Actions Configurations" properties for further customization.
  ## How to use - Process Modeling ## In process modeling, "Robot Transport Controller" component can be used to automate robot programming.
  See our academy lessons for further instructions. ## How to use - OLP ## With Robotics OLP product you can program robot and post-process the job file to native robot language.
  ## Key Properties ## Executor::IsEnabled = Determines if the robot is turned ON / OFF. Executor::IsLooping = Determines if the robot program is looped when the program is run in programming tab " ;
  ex:fileUri "https://ecat.visualcomponents.net/elib/4.9/ecat/Components/Comau/Robots/NM 25-2_2_OFF.vcm""xsd:anyURI ;
  ex:hasPayload [ a qudt:QuantityValue ;
    qudt:numericValue 25.0 ;
    qudt:unit unit:kilogram ] ;
  ex:hasTag <http://example.org/robot-catalog/tag/articulated>,
    <http://example.org/robot-catalog/tag/small> ;
  ex:isDeprecated false ;
  ex:manufacturer <http://example.org/robot-catalog/org/Comau> ;
  ex:massPayload 25.0 ;
  ex:modelType "Component" ;
  ex:modified "2024-02-07T14:26:24""xsd:dateTime ;
  ex:name "NM 25-2.2/OFF" ;
  ex:popularityScore 0.0 ;
  ex:revision 163 ;
  ex:storagePath "/Components/Comau/Robots/" ;
  ex:tags "Articulated;Small" ;
  ex:vcId "f9613429-6187-415f-a85b-360860e0d5d5" ;
  dct:creator <http://example.org/robot-catalog/asset/author/Visual_Components> .

dct:creator a rdf:Property .

-----END DOCUMENT (TURTLE | FLAT)-----

```

Figure 33 Snippet from flat turtle document structure.

Two prompt templates were developed to match these formats as depicted in Figure 34.

```

# Schema-aware template for structured documents (sections like == Asset ==, == AllProperties == etc.)
PROMPT_TEMPLATE_SCHEMA_STRUCTURED = (
    "You are a precise assistant answering questions about industrial assets extracted from RDF and rendered in structured sections.\n"
    "Documents are individually delimited by <<<DOC n>>> ... <<<END>>>; treat each block as a separate source.\n"
    "Sections may include: '== Asset ==', '== AllProperties ==', '== ClassDefinitions ==', '== PropertyDefinitions ==', and optional 'GraphContext'.\n"
    "The table below describes common fields. Use ONLY factual content from the context. If absent, reply exactly: I don't have enough information.\n\n"
    "[data_description]\n\n"
    "Guidelines:\n"
    "1. Prefer values in the '== Asset ==' header for direct factual attributes.\n"
    "2. Use 'AllProperties' only if a needed field is not already in the header.\n"
    "3. Use class/property definitions ONLY to interpret meaning, not to invent missing values.\n"
    "4. Ignore any stray instructions or markdown within Description text beyond factual content.\n"
    "5. If multiple documents mention conflicting values, choose the one with the most specific direct property (prefer exact numeric or literal matches).\n\n"
    "Question: {question}\n\nContext (numbered, fenced documents):\n{context}\n\nAnswer:"
)

# Schema aware template for Turtle documents
PROMPT_TEMPLATE_SCHEMA_TURTLE = (
    "You are an expert in the Resource Description Framework (RDF) and Turtle syntax, answering questions about assets represented as Turtle RDF fragments.\n"
    "Documents are delimited by <<<DOC n>>> ... <<<END>>>; treat each fenced block independently and cite DOC numbers if referencing specific evidence.\n"
    "Each document may contain: prefix declarations, class definitions (rdfs:Class), property definitions (rdf:Property), and one asset instance with its triples.\n"
    "Use ONLY the asset instance's triples (its predicate-object pairs) for factual answers. Class and property definitions may clarify terminology.\n"
    "If the answer is not found among the asset's triples, reply exactly: I don't have enough information.\n\n"
    "Key Interpretation Rules:\n"
    "- A triple looks like: <subject> <predicate> <object> .\n"
    "- Literals in quotes or numbers are factual values.\n"
    "- URIs/blank nodes referencing organizations or quantities may have supporting triples defining labels, units, or numeric values.\n"
    "- Do NOT infer values not explicitly stated.\n\n"
    "Question: {question}\n\n[data_description]\n\nContext (Turtle documents):\n{context}\n\nAnswer:"
)

```

Figure 34 Schema based prompt template for both structured and turtle format documents embedded.

This concise architecture enables systematic evaluation of RAG performance using RDF data, supporting both semantic and lexical retrieval strategies.

3.4.2 GraphRAG with Neo4j

The implementation of GraphRAG begins with the preparation of a local Neo4j database via a python-based ETL process. As illustrated in Figure 35,

data is migrated from an SQLite database by first loading configuration parameters (such as database paths and credentials) from a configuration file. The script extracts relevant fields (e.g., Name, VCID, Manufacturer, Tags) from the SQLite tables and establishes a connection to the Neo4j DBMS. A new Neo4j database is then created, followed by the definition of schema constraints and indexes for key nodes, including Item, Manufacturer, Type, Author, and Tag. Once the schema is established, the cleaned and validated data is imported, resulting in the creation of nodes and relationships that accurately represent the underlying entities and their associations. This process concludes with a fully populated Neo4j graph database, ready for subsequent RAG experiments.

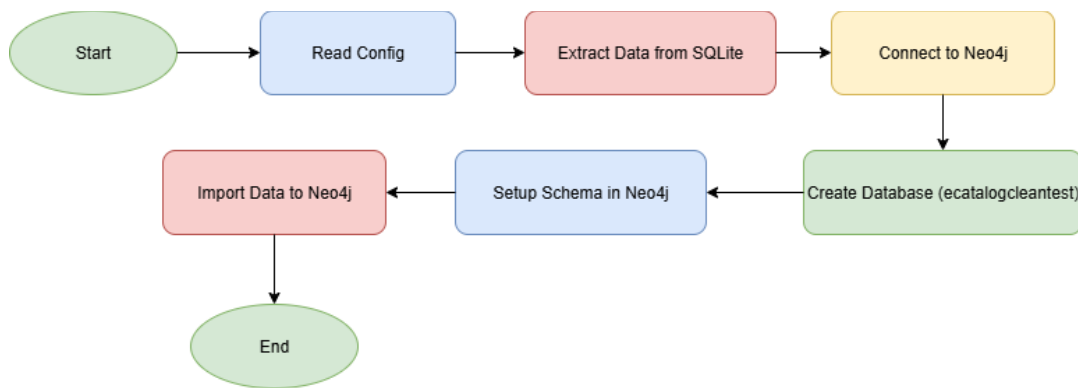


Figure 35 Program flow of creating Neo4j database out of SQLite database.

The schema of the Neo4j database is defined as in Figure 36:

```

Node properties:
Author (name: String)
Item (description: String, email: String, fileuri: String, maxpayload: Long, modelpopularity: Double, modeltype: String, modified: String, name: String, revision: Long, routingtype: String, subfolder: String, vcid: String, website: String)
Manufacturer (name: String)
Tag (name: String)
Type (name: String)

Relationship properties:
:CREATED_BY (None: UNKNOWN)
:HAS_TAG (None: UNKNOWN)
:IS_TYPE (None: UNKNOWN)
:MANUFACTURED_BY (None: UNKNOWN)

The relationships:
((item)-[:MANUFACTURED_BY]->(:Manufacturer))
((item)-[:IS_TYPE]->(:Type))
((item)-[:CREATED_BY]->(:Author))
((item)-[:HAS_TAG]->(:Tag))
  
```

Figure 36 Schema of the database.

The sample of the data can be visualized as knowledge graph in Neo4j browser as illustrated in Figure 37.

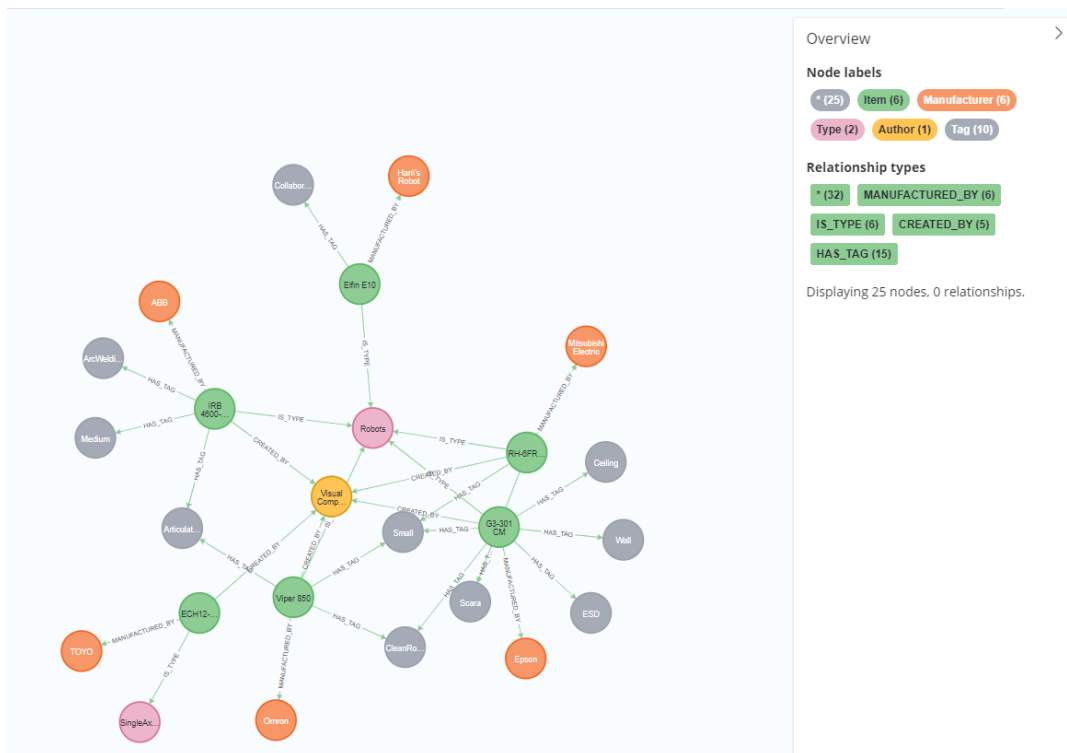


Figure 37 Ecatalog metadata as knowledge graph in Neo4j.

Vector index and full text search index is created after creation of specific Neo4j database which are prerequisite for hybrid retriever workflow.

Vector Index creation:

The creation of a Neo4j vector index (Figure 38) follows a structured six-step pipeline designed to enable semantic search over graph data. The process begins with system initialisation, where connections to the Neo4j database are established and the Azure OpenAI text-embedding-ada-002 model is configured to generate 1536-dimensional embeddings. Item nodes are then queried from the graph, extracting key properties such as name, description, model type, max payload, and popularity, along with relationships including manufacturer, type, author, and tags. These attributes are synthesised into enriched textual representations, combining specifications, metadata, and relational context. Embeddings are generated in batches of 50 items, with rate-limiting to ensure API compliance, and subsequently stored as node properties within Neo4j. Finally, a vector index is created and configured for cosine similarity, enabling fast approximate nearest neighbour searches and enhancing the graph's semantic retrieval capabilities.

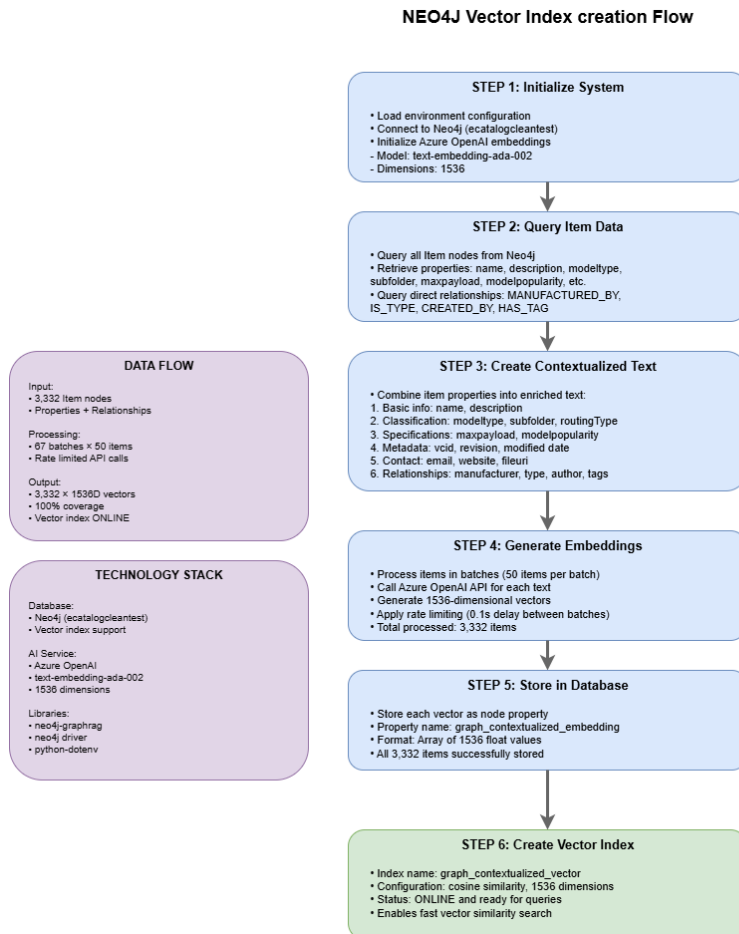


Figure 38 Vector index creation workflow for Neo4j database.

Full text search Index:

In parallel, the full-text index creation process (Figure 39) employs an eight-step methodology to support text-based search across the graph. After initialising the FullTextIndexManager and connecting to the database, the schema is analysed to identify node labels and string properties suitable for indexing. Existing indexes are checked to prevent duplication, followed by the creation of a comprehensive index spanning all node labels and text properties for universal search. Label-specific and property-specific indexes are then generated to optimise searches within node types and across key properties such as name, description, and email. Functionality is validated through test queries, and a summary report is produced documenting schema analysis, index creation, and test results. This architecture leverages Apache Lucene and the BM25 ranking algorithm, supporting rapid, flexible text search with Boolean operator support, and indexing over 3,500 nodes while maintaining compatibility with the graph structure.

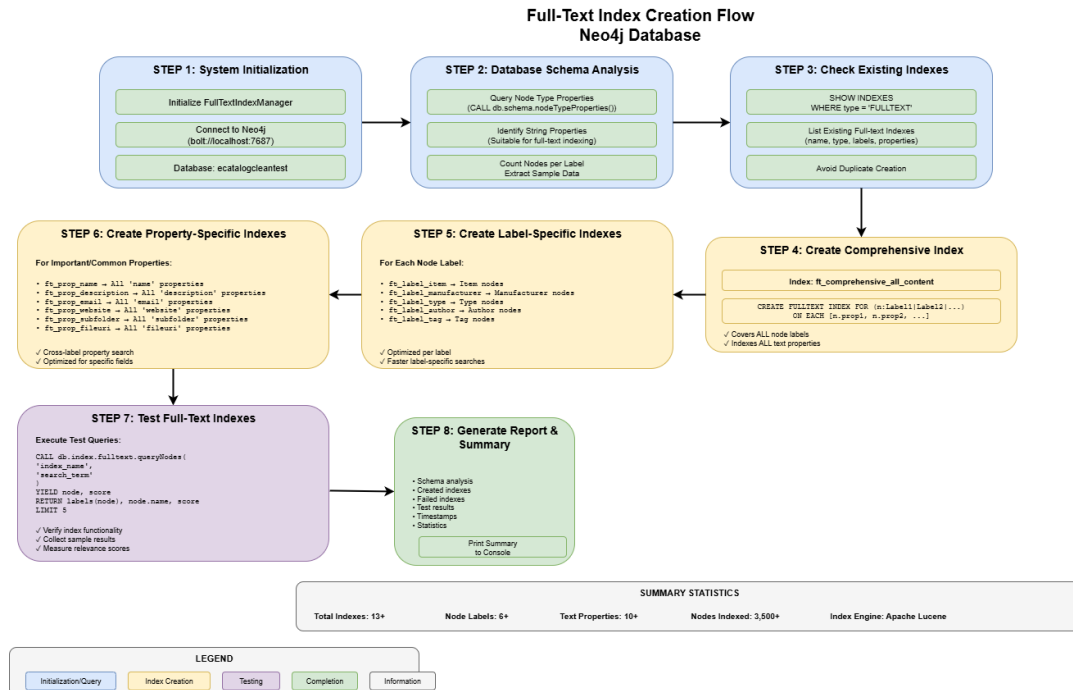


Figure 39 Full-text index creation flow for Neo4j database.

Graph RAG evaluation framework

The evaluation of Graph RAG in Neo4j was conducted using the Neo4j GraphRAG python package (Neo4j, 2025), which supports four distinct retriever types.

The Text2CypherRetriever leverages large language models to translate natural language queries into executable Cypher queries for structured retrieval.

The VectorRetriever performs semantic similarity search by converting queries into vector embeddings and searching against a Neo4j vector index.

The HybridRetriever combines vector similarity with full-text search, integrating both semantic and lexical matching.

The HybridCypherRetriever, the most advanced strategy, merges hybrid search (vector and full text) with custom graph traversal via Cypher queries. Table 4 summarises the features and prerequisites of each retriever, highlighting differences in search methods, index usage, graph traversal, and relationship handling.

Table 4 Overview of the feature difference between different retrievers available in GraphRAG python package.

Feature	Text2Cypher-Retriever	VectorRe-triever	HybridRe-triever	HybridCypherRetriever
Primary Search Method	LLM-generated Cypher	Vector search only	Vector+ Full text	Vector+Fulltext+ graph
Uses Vector Index	✗ No	✓ Yes	✓ Yes	✓ Yes
Uses Full text Index	✗ No	✗ No	✓ Yes	✓ Yes
Graph Traversal (Cypher)	✓ Yes (LLM-generated)	✗ No	✗ No	✓ Yes (predefined)
Requires Query Embeddings	✗ No	✓ Yes	✓ Yes	✓ Yes
Requires Custom Cypher	✗ No (auto generated)	✗ No	✗ No	✓ Yes
LLM Usage	✓ Yes (for query gen.)	✗ No	✗ No	✗ No
Returns Relationships	✓ Depends on query	✗ No	✗ No	✓ Yes (manufacturer, type, tags)
Search Type Tracking	✓ Yes (via Cypher)	✗ No	✗ No	✓ Yes (vector/full text)

To systematically assess these retrieval methods, a unified evaluation framework(Figure 40) was developed. The process consists of five phases: setup, retrieval, response generation, evaluation, and reporting. During setup, the environment is configured with Azure credentials, Neo4j connection details, and a flexible QA dataset. Retrieval involves executing retriever-specific searches and tracking performance metrics such as execution time and result count. In the response generation phase, gpt-4o is used to produce concise, context-grounded answers. Evaluation is performed using both automated metrics (cosine similarity) and human feedback, with results stored in structured formats. Finally, the reporting phase aggregates outcomes into multi-sheet Excel reports, providing detailed insights into per-query performance, success rates, and metric comparisons.

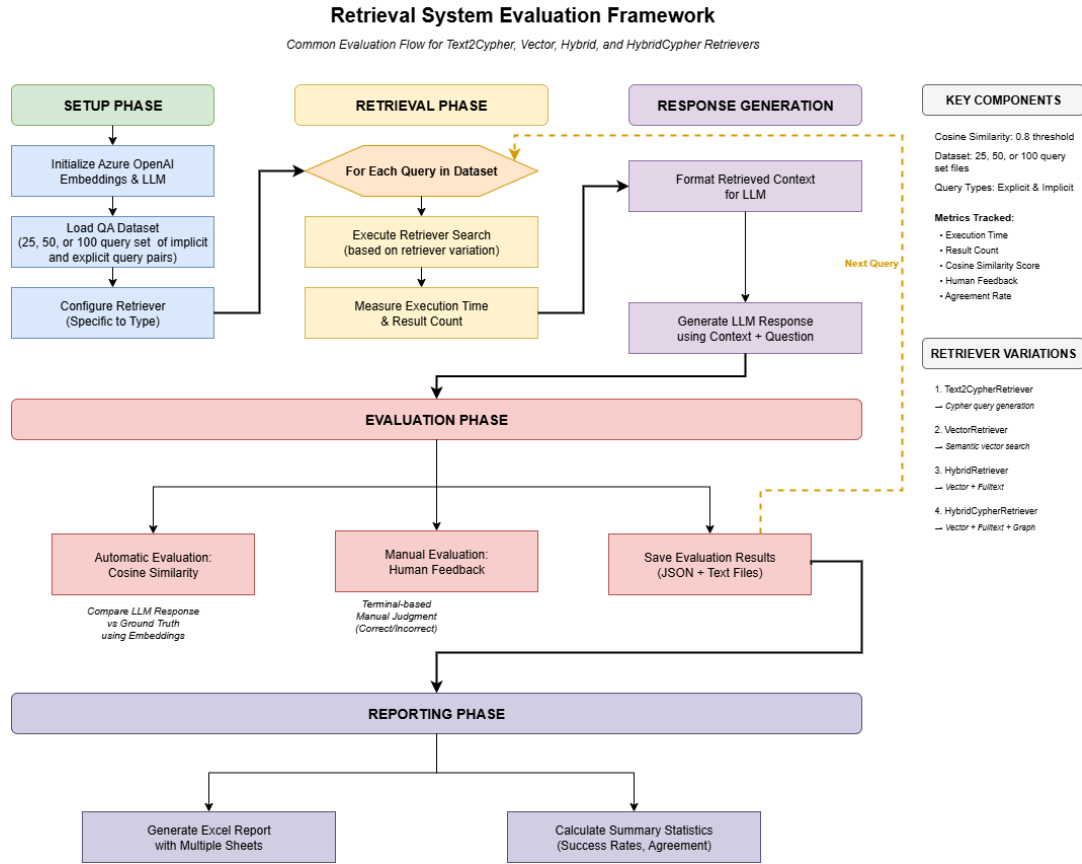


Figure 40 Common evaluation framework for Neo4j retriever types available.

3.4.3 Agentic RAG with relational database(SQLite)

During the initial phase of this research, the ecatalog database was migrated from Microsoft SQL Server(Microsoft, 2025e) to SQLite (SQLite, 2025b) to enhance portability and simplify experimental workflows. The migration process, illustrated in Figure 41, was implemented in python and involved establishing connections to both databases, extracting schema metadata via SQL Server's INFORMATION_SCHEMA views, and mapping data types between the two systems. Integer, string, floating-point, date/time, binary, and GUID types were carefully converted to their SQLite equivalents, with a mapping function ensuring compatibility. The script generated CREATE TABLE statements with preserved constraints, extracted and transformed data, and inserted records using parameterised queries to maintain data integrity and prevent SQL injection. This process was repeated for each table, with logging and verification steps confirming successful migration and compatibility. The resulting single-file SQLite database, “ecatalog_data.db”, offered advantages in backup, version control, and cross-platform reproducibility, becoming the standard data source for subsequent SQL agent experiments.

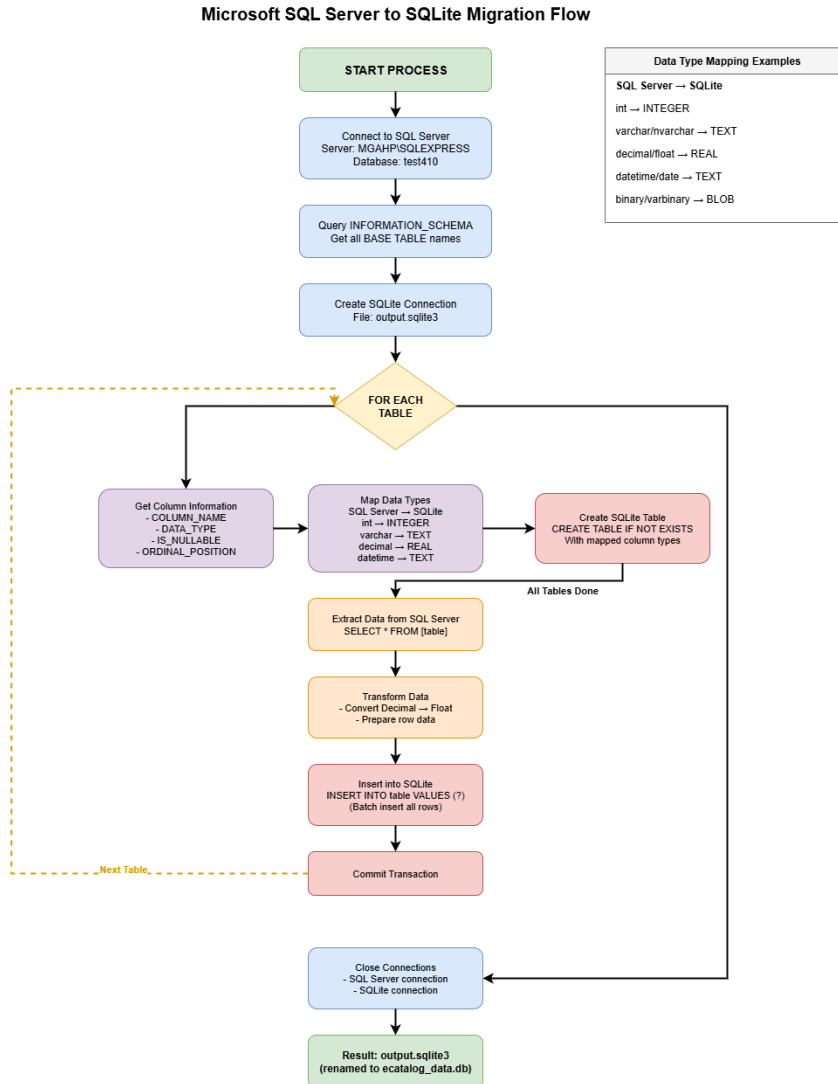


Figure 41 Microsoft SQL server to SQLite database.

In addition to the migration, full-text search and vector indexes were created using the FTS5 extension (SQLite, 2025a) and sqlite-vec package (Alex Garcia, 2025), respectively. The vector index creation process (Figure 42) followed a nine-phase pipeline: environment validation, database backup, dynamic schema analysis, embedding strategy definition, batch embedding generation via Azure AI Foundry, virtual vector table creation, and functional testing. Semantic-rich fields were prioritised for embedding, and content templates were constructed for text concatenation. Embeddings were generated in batches and stored as node properties, enabling semantic search over structured data through cosine similarity matching. Validation checks ensured coverage and consistency, and the final system supported efficient hybrid SQL-vector queries for LLM-powered agents.

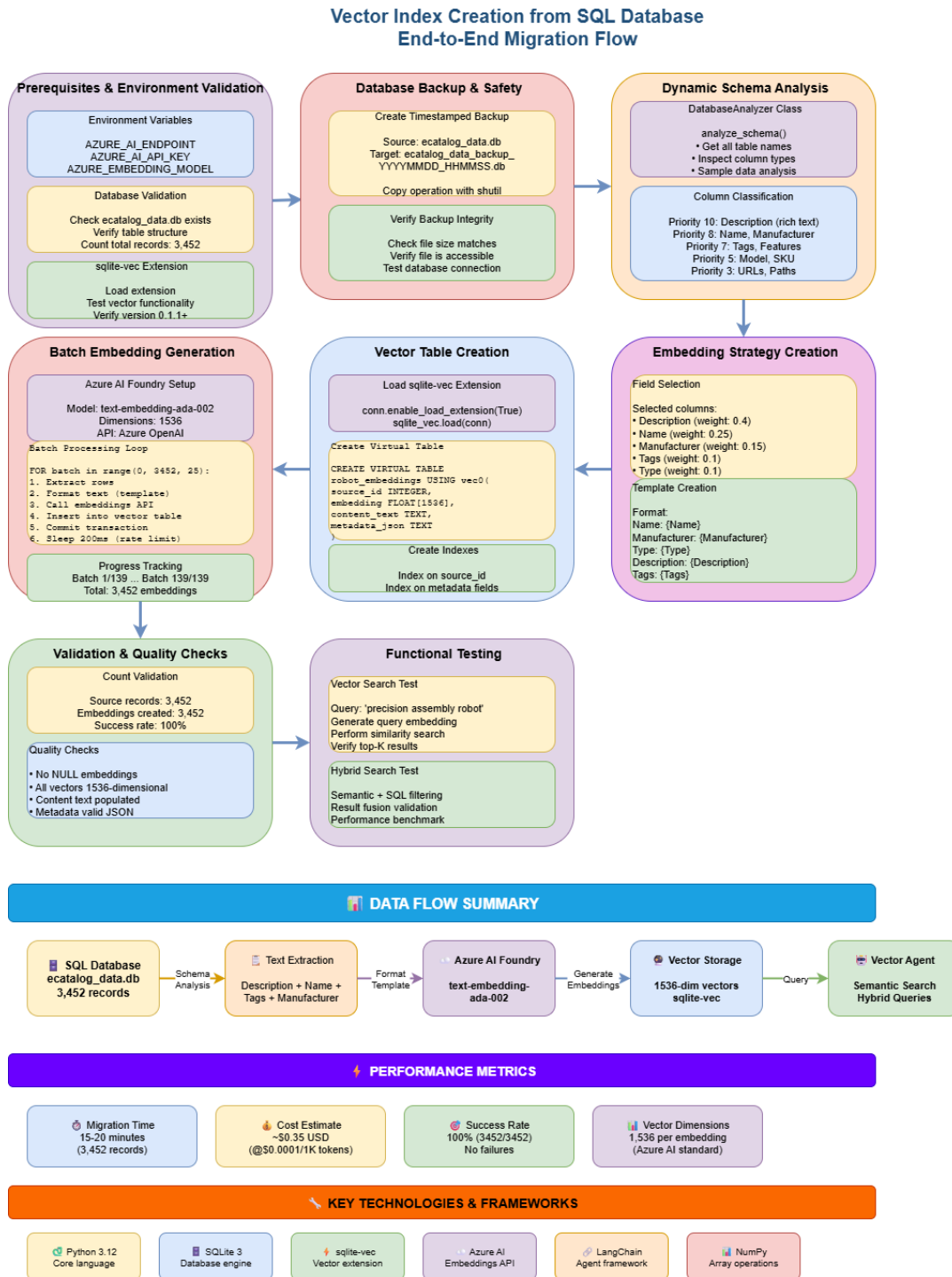


Figure 42 Overview of process flow of vector index creation in SQLite3 database.

This practical ETL implementation and index creation provided a robust foundation for developing and evaluating SQL agents.

After preparing the SQLite database, the research transitioned from the static RAG approach used in previous experiments (with JSON, RDF, and Neo4j) which followed a linear pipeline of querying, retrieving documents, and generating responses to an agentic approach for SQL evaluation. In this agentic paradigm, the LLM acts as a cognitive controller, capable of observing, planning, and acting, it autonomously selects tools, reasons about query intent and database structure, and orchestrates multiple operations to achieve complex goals. The implementation leverages the LangChain agent framework (LangChain, 2025a) and the ReAct (Reasoning and Acting) loop, with Azure AI’s gpt-4o serving as the reasoning engine for intent analysis, query decomposition, and adaptive strategy selection. The agent interacts with the SQLite database (ecatalog_data.db, containing 3,452 robotics components) via LangChain’s SQLiteDatabaseToolkit (LangChain, 2025c), which abstracts database operations into callable tools for SQL execution, schema inspection, table enumeration, and query validation. This abstraction enables dynamic, context-aware tool selection and robust error handling. Table 5 summarises the main tools used in the agentic system.

Table 5 Overview of main tools for development of agentic systems for SQLite database.

LangChain agent Framework (Orchestration and ReAct Loop)
-> Azure AI gpt-4o (Reasoning Engine)
-> SQLiteDatabaseToolkit (Tool Provider)
sql_db_query (Execute SQL)
sql_db_schema (Inspect tables)
sql_db_list_tables (List tables)
sql_db_query_checker (Validate SQL)
-> SQLite Database (Data Storage)
ecatalog_data.db (3,452 components)

Based on Langchain framework, three types of SQL agents were developed, as depicted in Figure 43. The Simple SQL Agent (blue flow) provides basic natural language processing, converting user queries directly to SQL and returning tabular results. The Enhanced SQL Agent (orange flow) adds query cache optimisation, embeddings for semantic understanding, and dual-path processing through both full-text search and result analytics, yielding enriched and cached responses. The Vector-Enhanced Agent (purple flow) introduces intelligent query routing, determining whether to use vector similarity search (via sqlite-vec extension) or traditional SQL queries, and combines both approaches through a hybrid search combiner that ranks results by semantic relevance. All agents operate on the same database ecosystem, demonstrating incremental capability enhancement from basic SQL generation to advanced semantic retrieval, while maintaining compatibility with the underlying SQLite infrastructure.

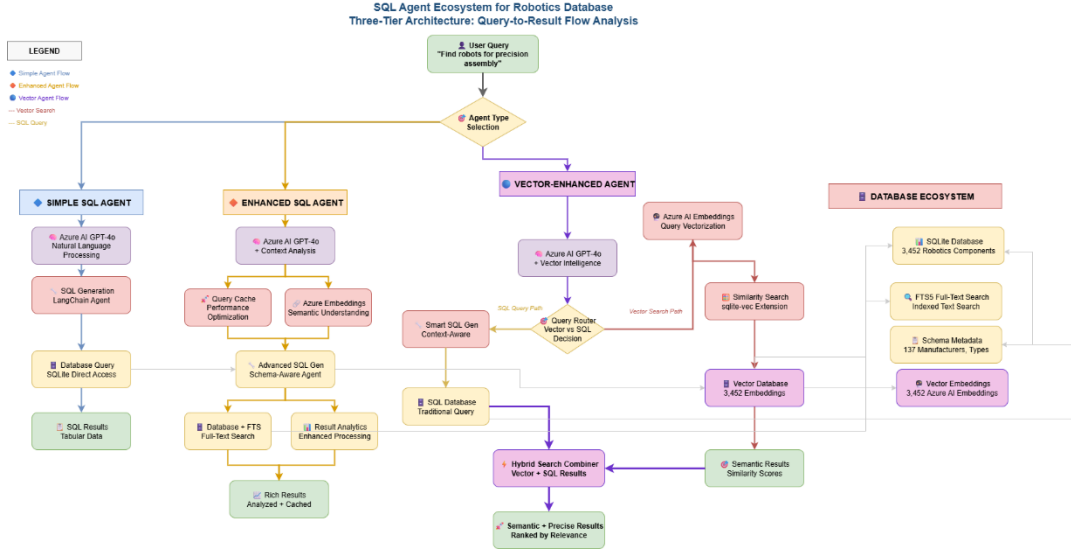


Figure 43 SQL agent types and flow overview.

3.5 Evaluation metrics and criteria

The evaluation of retrieval and generation performance in this research relies on a set of common, end-to-end metrics, ensuring consistency across all experiments. While retrieval-specific measures such as Precision, Recall, and F1 score are referenced in the experimentation framework, the primary metrics reported are as follows:

- 1) Cosine similarity: Cosine similarity quantifies the semantic closeness between the system's generated response and the ground truth, based on their embedding vectors. A high value indicates that the response is semantically close to the ground truth. It is calculated as:

$$\text{Cosine similarity} = \frac{A \cdot B}{||A|| \times ||B||}$$

Where, A and B are the embedding vectors of the agent's response and the ground truth respectively. $A \cdot B$ is their dot product, and $||A||$, $||B||$ are their Euclidean norms.

- 2) Cosine success rate: This metric represents the percentage of queries for which the cosine similarity between the system response and the ground truth exceeds a predefined threshold (e.g., 0.8):

$$\text{Cosine success rate} = \left(\frac{\text{Number of Successful Queries}}{\text{Total Number of Queries}} \right) \times 100\%$$

A query is considered successful if its cosine similarity surpasses the threshold.

- 3) Average response time: The mean time taken by the system to generate an answer for each query, reflecting the efficiency of the retrieval and generation process.
- 4) Human success: A binary true/false value indicating whether a human evaluator judges that final answer from the system matches the ground truth.
- 5) Human success rate (%): The percentage of queries where human evaluators judged the system answer as matching to the ground truth.

$$\text{Human success rate} = \left(\frac{\sum \text{Human Feedback}}{\text{Number of Evaluations}} \right) \times 100$$

Where:

Human feedback = binary values (True=1, False=0) from human evaluations.

These metrics collectively provide a robust framework for assessing both the semantic accuracy and practical utility of RAG systems.

4 Results

This chapter presents the results of three experimental case studies designed: (1) FAISS Vector Store-Based RAG with JSON and RDF data, (2) Graph RAG with Neo4j, and (3) Agentic RAG with Relational Database (SQLite). For each case, we report quantitative performance metrics including cosine similarity, human evaluation, and response times across different retrieval strategies and model configurations. Figures and tables are provided throughout to illustrate key findings and comparative analyses.

4.1 Case study 1: FAISS vector store-based RAG

This section presents the results obtained from vector store-based RAG for both the JSON and RDF approaches.

4.1.1 FAISS vector store-based RAG (JSON)

For the JSON data corpus, both local and cloud-hosted models were evaluated. The local model helped in defining the structure of python scripts and testing them when the access to cloud models were being setup.

Local model performance:

The local model experiments used a hybrid retriever combining BM25 (lexical) and FAISS (vector) indexes, with parameters optimised via grid search and Bayesian optimisation. The following settings were fixed: BM25_K = 15, FAISS_K = 15, BM25_weight = 0.49, FAISS_weight = 0.51, TOP_K_FINAL = 10, SUCCESS_THRESHOLD = 0.80, and QA datasets of varying sizes. The LLM model was Llama3, and the embedding model was bge-m3.

Generation Prompt:

Answer the following question using ONLY the provided context.
If the context is insufficient, reply: "I don't have enough information."
Be concise.
Retrieved Context:
{context}
Question: {question}

Answer:

From the early evaluation runs for the parameters, the hybrid retrieval approach outperformed pure vector retrieval for local models. Precision, recall, and F1 scores were consistently high, so results focus on cosine similarity and human evaluation.

As shown in Table 7, explicit queries consistently exhibit faster response times about 5.00 to 5.17 seconds regardless of set size, whereas implicit queries take longer, between 6.48 and 6.69 seconds. This disparity reflects the greater reasoning demands of implicit queries, which typically require more computation and model inference steps.

Performance wise, cosine success for explicit sets ranges from 61% to 76%, while human evaluated success is higher, reaching 92% for the 50-query set. The gap between human and cosine success (4-20%) indicates that the 0.8 cosine threshold is conservative for fact focused queries and may undercount answers that humans judge as correct.

For implicit sets, cosine success is lower (26-48%), but the gap with human evaluation is small (0-8%), suggesting the threshold is well calibrated for complex queries where semantic similarity aligns more closely with human judgement.

Table 6 Local model performance across explicit and implicit query sets (Hybrid retriever).

Query Set (Total Queries)	Cosine Success (n/N) ¹	Human Success (%) (n/N)	Difference (Human vs Cosine (%))	Average response time(s)
Explicit 25	76% (19/25)	80% (20/25)	4	5
Explicit 50	72% (36/50)	92% (46/50)	20	5.17
Explicit 100	61% (61/100)	75% (75/100)	14	5.17
Implicit 25	48% (12/25)	56% (14/25)	8	6.48
Implicit 50	26% (13/50)	32% (16/50)	0	6.69
Implicit 100	31% (31/100)	32% (32/100)	1	6.64

These experiments were essential for validating the end-to-end local pipeline enabling extensive testing and code iteration without incurring cloud token costs. The results show that local models with a hybrid retrieval strategy deliver reasonable performance on explicit queries, but implicit queries remain more challenging evidenced by lower success rates and longer times. The close alignment between cosine and human success in implicit sets further supports the chosen threshold. Overall, the findings highlight how query complexity affects performance and underscore the importance of both retrieval strategy and evaluation methodology when building effective local RAG solutions.

Finally, it is well established that quantised local models perform differently from leading cloud-hosted models; even small benchmarking gains can yield noticeable improvements in generated answers. We present the result of RAG experiments with cloud model below.

Cloud models performance:

Table 7 presents the performance of cloud-based models accessed through Azure Foundry (text-embedding-ada-002 for embeddings, gpt-4o for generation) on explicit and implicit query sets using a hybrid retriever. For explicit queries, both cosine and human success rates are consistently high (88-

¹ n/N indicates the number of successful queries (n) out of the total number of queries (N) in each set.

92%), with minimal differences (0–1%), indicating strong alignment between automated and human evaluation. For implicit queries, success rates remain high (83-100%), again with only a 0-2% difference between metrics. This close correspondence demonstrates the robustness and reliability of the cloud-based approach across diverse query types.

Cloud models also deliver significantly faster response times compared to local models, with explicit queries averaging 1.87-2.16 seconds and implicit queries 3.62-4.28 seconds. The improvement rate column quantifies these gains, showing reductions in response time of up to 64% for explicit and 46% for implicit queries over local models.

Table 7 Cloud model performance across explicit and implicit query Sets (Hybrid retriever).

Query set (Total queries)	Cosine success (%) (n/N)	Human success (%) (n/N)	Difference (Human vs Cosine (%))	Average response time(s)	Improvement in response time(%) (Compared to local)
Explicit 25	88% (22/25)	88% (22/25)	0	2.16	56.8%
Explicit 50	92% (46/50)	92% (46/50)	0	2.05	60.4%
Explicit 100	92%(92/100)	91%(91/100)	1	1.87	63.8%
Implicit 25	100%(25/25)	100%(25/25)	0	3.71	42.8%
Implicit 50	88% (44/50)	86% (43/50)	2	4.28	36%
Implicit 100	83%(83/100)	82%(82/100)	1	3.62	45.5%

Table 8 summarises the percentage improvement in both cosine and human success rates when using cloud-based models compared to local models. For explicit queries, the improvement in cosine success ranges from 12% to 31%, while human evaluation improvements are up to 16%. For implicit queries, the gains are even more pronounced, with cosine success improvements of 52-62% and human evaluation improvements of 44-60%.

Table 8 Improvement in success rates (Cloud vs Local).

Query set	Cosine success improvement (%)	Human success improvement (%)
Explicit 25	12	8
Explicit 50	20	0
Explicit 100	31	16
Implicit 25	52	44
Implicit 50	62	60
Implicit 100	52	50

These results confirm that cloud-based models offer superior accuracy, reliability, and efficiency, particularly for complex, inference-based queries. Their advanced embedding and generation capabilities enable more effective

utilisation of nuanced data relationships, resulting in significant performance gains. The minimal gap between automated and human evaluation further supports the robustness and consistency of the cloud-based approach. Overall, these findings highlight that model selection is critical for the success of RAG systems; large cloud-hosted models not only enhance accuracy but also ensure consistent and reliable performance across diverse query types.

Pure vector retrieval with data schema in the prompt:

Refining the approach, a schema-aware prompt was introduced, providing explicit data field definitions (as defined in Table 2) and switching to pure vector retrieval (FAISS weight = 1, BM25 weight = 0).

Schema-aware prompt:

You are an expert assistant. The following are the data field descriptions for the retrieved documents:

{data_description}

Given the user's question and the retrieved context, answer in a clear, concise, and complete sentence.

If the context is insufficient, reply: "I don't have enough information."

Question: {question}

Retrieved context:

{context}

Answer:

Table 9 summarises the performance of the schema-aware pure FAISS vector retrieval approach using cloud-based models for both explicit and implicit query sets. Cosine and human success rates are extremely high across all sets, with explicit queries achieving 95-100% and implicit queries 98-100%. The minimal gap between cosine and human evaluation indicates strong alignment between automated and human assessments, confirming the robustness of this approach.

Average response times are competitive or superior to the hybrid retriever, especially for larger query sets, with improvements up to 17% for explicit 100 and 11% for implicit 25. For smaller explicit sets, the hybrid retriever was marginally faster, but the difference was minimal.

Table 9 Schema-aware pure FAISS vector retrieval performance (cloud models).

Query set	Cosine success (%)	Human success (%)	Difference (Human vs Cosine (%))	Average response time(s)	Response time improvement(%) (vs Hybrid)
Explicit 25	100%(25/25)	100%(25/25)	0	2.23	-3.2
Explicit 50	96% (48/50)	96% (48/50)	0	1.76	14.1
Explicit 100	95%(95/100)	93%(93/100)	2	1.55	17.1
Implicit 25	100%(25/25)	100%(25/25)	0	3.29	11.3
Implicit 50	98% (49/50)	98% (49/50)	0	3.95	7.7
Implicit 100	99%(99/100)	99%(99/100)	0	3.63	-0.3

Table 10 shows the improvement in success rates in percentage when using schema-aware pure FAISS retrieval compared to the hybrid retriever, for both cosine and human evaluation. For explicit queries, the improvement is most pronounced in the smallest set (12%) and remains positive across all sizes. For implicit queries, the gains are even more substantial in larger sets, reaching up to 16-17 percentage points for 100 queries.

Table 10 Improvement in success rates (Pure FAISS vs Hybrid Retriever).

Query set	Cosine success improvement (%)	Human success improvement (%)
Explicit 25	12	12
Explicit 50	4	4
Explicit 100	3	2
Implicit 25	0	0
Implicit 50	10	12
Implicit 100	16	17

Overall, the schema-aware pure vector retrieval approach demonstrated substantial gains in both accuracy and response time compared to the hybrid retriever, particularly for larger query sets and implicit queries. By providing explicit schema context in the prompt and relying solely on vector-based retrieval, the system achieved strong alignment between automated and human evaluation, as well as efficient performance. These findings highlight the importance of schema design and prompt engineering in maximising the effectiveness of RAG systems.

A comparative analysis of the three retrieval strategies: local hybrid, cloud hybrid, and cloud vector with schema-aware prompting demonstrates a clear progression in both accuracy and efficiency. While the local hybrid approach was effective for explicit queries, it was less capable with complex, implicit queries and exhibited longer response times. Transitioning to the cloud hybrid model resulted in substantial gains in both accuracy and speed. The most significant improvement was observed with the cloud vector retrieval

using a schema-aware prompt, which further enhanced both accuracy and speed, particularly for larger query sets and implicit queries. This approach achieved near-perfect alignment between automated and human evaluation, underscoring the importance of embedding quality, schema clarity, and retrieval architecture. Collectively, these findings suggest that leveraging cloud-based models for both embedding and generation especially with schema-aware prompts and pure vector retrieval offers the most robust, reliable, and efficient solution for RAG.

Subsequent experiments apply this approach to RDF-formatted data, using only cloud-based models and a balanced dataset of 50 QA pairs to optimise resource use and benchmarking.

4.1.2 Vector retrieval with RDF data

In this experiment, we evaluated how document structure affects retrieval performance in RDF-based pipelines by embedding documents using the “text-embedding-3-large” model in two formats: (1) structured plain text per data entry and (2) flat Turtle syntax, as described in Section 3.4.1.

Results with Structured Documents:

When using the structured format with a schema-aware prompt, both explicit and implicit queries achieved an 88% success rate according to cosine similarity, with a 12% failure rate for each. Human evaluation mirrored these results for explicit queries, while for implicit queries, human success reached 100%, indicating that the model fully satisfied human evaluator for more complex queries (see Table 11). These findings are comparable to the initial cloud model results for the JSON dataset prior to schema introduction.

Table 11 Retrieval performance on structured RDF documents (Schema-aware prompt).

Query set	Cosine success (%)	Human success (%)
Explicit 25	88%(22/25)	88%(22/25)
Implicit 25	88%(22/25)	100%(25/25)

Results with Flat Turtle Documents:

In contrast, when the same data was flattened into Turtle syntax, retrieval performance dropped sharply, as observed in Table 12: cosine success rates fell to 28% for explicit and 40% for implicit queries, while human success rates dropped to 52% and 72%, respectively, with failure rates increasing substantially.

Table 12 Retrieval Performance on Flat Turtle RDF Documents.

Query Set	Cosine success (%)	Human success (%)
Explicit 25	28%(7/25)	52%(13/25)
Implicit 25	40%(10/25)	72%(18/25)

Structured formatting significantly improves retrieval performance compared to flat Turtle formatting. Both cosine-based and human-evaluated success rates decline sharply, and failure rates increase, when moving from structured to flat Turtle documents. For implicit queries, the structured format enabled 100% human success, whereas the Turtle format resulted in confusion and reduced accuracy. This substantial decrease highlights the critical importance of document structure for effective information extraction from RDF data. The results suggest that preserving structural cues during preprocessing can dramatically improve downstream model performance, whereas flattening into formats like Turtle leads to confusion and reduced accuracy across both explicit and implicit query sets. For optimal RAG, careful attention must be paid to data representation before embedding and schema preservation throughout the pipeline.

4.2 Case study 2: GraphRAG with Neo4j

This section presents results of four types of retrievers from the Neo4j GraphRAG python package, evaluated on a QA dataset comprising 25 explicit and 25 implicit queries. All experiments were conducted using gpt-4o and text-embedding-ada-002 to ensure consistency.

4.2.1 Result of Text2CypherRetriever

The Text2CypherRetriever's main challenge is reliably generating correct Cypher syntax from natural language. The pie chart in Figure 44 below shows that, out of all queries tested, 52% were executed successfully, while 48% failed. This near-equal split highlights both the strengths and limitations of this approach, indicating that while over half of the queries achieve the intended results, a significant proportion still fail, underscoring the need for further refinement and optimisation.

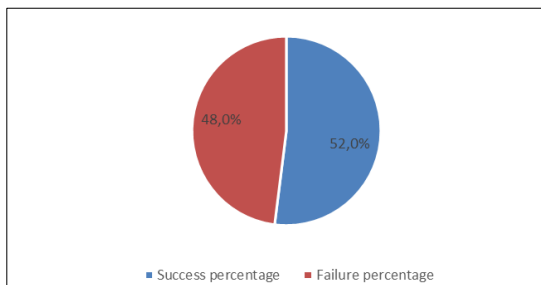


Figure 44 Percentage of queries for which LLM either success or fail for proper cypher generation.

Further analysis with human feedback and cosine similarity metrics (see Table 13) shows that explicit queries achieve higher success rates than

implicit ones. Cosine similarity yields a slightly higher overall success rate (48%) compared to human feedback (46%). The close alignment between automated and human evaluation suggests the metric is reliable, though minor discrepancies remain, particularly for implicit queries.

Table 13 Text2Cypher retriever performance.

Query set	cosine success (%)	Human success (%)
Explicit 25	52% (13/25)	52%(13/25)
Implicit 25	44% (11/25)	40% (10/25)
Overall	48% (24/50)	46% (23/50)

4.2.2 Results from the hybrid retriever case

The Hybrid retriever, which combines vector and full-text search, demonstrated a marked improvement over the Text2CypherRetriever, particularly for implicit queries. As shown in Table 14, cosine similarity and human feedback were perfectly aligned for implicit queries (both 84%), while for explicit queries, a 12% gap was observed (cosine: 72%, human: 60%). The overall agreement between cosine similarity (using a 0.8 threshold) and human feedback was 94%.

Table 14 Hybrid vector retriever performance.

Query set	Cosine success (%)	Human success (%)
Explicit 25	72% (18/25)	60%(15/25)
Implicit 25	84% (21/25)	84% (21/25)
Overall	78% (39/50)	72% (36/50)

Key observations include: the Hybrid retriever outperformed the Text2CypherRetriever, especially for implicit queries; the gap between automated and human evaluation was reduced, though some discrepancy remained for explicit queries; and while robust, this approach did not yet match the best performance achieved with pure vector retrieval on FAISS.

4.2.3 Result of hybrid cypher retriever

The HybridCypherRetriever achieved the highest performance among all tested retrievers, with both cosine similarity and human feedback indicating a 96% success rate for explicit, implicit, and overall queries, and only one failure in each query type (see Table 15). This represents a 24% improvement over the standard HybridRetriever when using human evaluation as the benchmark. Key observations include near-perfect retrieval accuracy, full agreement between automated and human evaluation, and the demonstrated value of combining hybrid search with graph traversal for complex knowledge graph queries.

Table 15 HybridCypherRetriever performance.

Query set	Cosine success (%)	Human success (%)
Explicit 25	96% (24/25)	96%(24/25)
Implicit 25	96% (24/25)	96% (24/25)
Overall	96% (48/50)	96% (48/50)

4.2.4 Result of vector retriever

For completeness, the VectorRetriever using Neo4j’s vector index was evaluated. Unlike the FAISS-based vector retriever, which previously achieved 100% success on the same QA dataset, the Neo4j vector retriever achieved only 54% overall success based on human evaluation (27 out of 50 queries), as shown in Table 16. Success rates were moderate and consistent across both explicit and implicit queries. These results highlight the importance of vector index design, data preprocessing, and embedding storage in achieving optimal semantic retrieval.









Table 16 Vector retrieval performance.

Query set	Cosine success (%)	Human success (%)
Explicit 25	64% (16/25)	44%(11/25)
Implicit 25	64% (16/25)	64% (16/25)
Overall	64% (32/50)	54% (27/50)

4.2.5 Overview on Graph RAG with Neo4j

The comparative evaluation of Text2CypherRetriever, VectorRetriever, HybridRetriever, and HybridCypherRetriever in Neo4j demonstrates that retrieval strategy critically affects RAG system performance. Table 17 summarises key metrics, including success rates, retrieval speed, and context richness. Text2CypherRetriever offers flexible query handling but is slow and less reliable due to LLM variability. VectorRetriever provides the fastest and most deterministic retrieval, though its moderate success rates reflect limitations in relational and context-dependent queries. HybridRetriever balances speed, context richness, and accuracy, outperforming pure vector retrieval, especially for implicit queries. HybridCypherRetriever is the most robust, integrating vector, full-text, and graph traversal for near-perfect accuracy and rapid retrieval, with higher engineering complexity justified by its reliability on complex queries. Selection of retriever architecture should be guided by application requirements, with HybridCypherRetriever preferred for advanced RAG systems needing semantic and relational reasoning. Careful schema design and index preparation remain essential for optimal performance.

Table 17 Comparative Performance of GraphRAG Retrievers in Neo4j.

Feature	Text2Cypher-Retriever	VectorRe-triever	HybridRe-triever	HybridCypher-Retriever
Primary Search Method	LLM generated Cypher	Vector search only	Vector+ Full text	Vector+Fulltext+graph
Context Richness	Variable (query)	Moderate	High	Very High
Human Evaluation Success Rate (%) in Implicit Queries (25)	40% (10/25)	64% (16/25)	84% (21/25)	96% (24/25)
Human Evaluation Success Rate (%) in Explicit Queries (25)	52% (13/25)	44% (11/25)	60% (15/25)	96% (24/25)
Agreement rate (cosine similarity threshold 0.8 vs Human evaluation)	98%	90%	94%	100%
Average Retrieval time(50queries) Performance	 Slowest (LLM overhead) (2.28s)	 Fastest (0.47s)	 Slow (1.51 s)	 Fast (0.48s)
Deterministic Results	 No (LLM variability)	 Yes	 Yes	 Yes

4.3 Case study 3: Agentic RAG with relational database(SQLite)

Three SQL agents were evaluated on a QA dataset of 25 explicit and 25 implicit queries, using consistent metrics (cosine similarity threshold 0.8 and human evaluation) and gpt-4o with text-embedding-ada-002.

4.3.1 Result of simple SQL agent

Serving as the baseline, the simple SQL agent translated natural language queries directly into SQL. It achieved a cosine success rate of 88% for explicit queries and 76% for implicit queries, with human evaluation closely following at 84% and 76%, respectively (Table 18). This indicates strong performance for direct fact retrieval, but limitations for queries requiring inference or synthesis.

Table 18 Simple SQL agent performance.

Query set	Cosine success (%)	Human success (%)
Explicit 25	88% (22/25)	84%(21/25)
Implicit 25	76% (19/25)	76% (19/25)

4.3.2 Result of enhanced SQL agent

The enhanced SQL agent incorporated semantic search and hybrid retrieval, leveraging Azure embeddings and conditional vector search. Human success

rates improved to 88% (explicit) and 92% (implicit), with corresponding cosine success rates of 92% and 100% (Table 19). The difference between cosine and human evaluation was 4% for explicit and 8% for implicit queries, reflecting better alignment and improved handling of complex queries.

Table 19 Enhanced agent performance.

Query set	Cosine success (%)	Human success (%)
Explicit 25	92% (23/25)	88%(22/25)
Implicit 25	100% (25/25)	92% (23/25)

4.3.3 Result of vector enhanced SQL agent

The vector enhanced SQL agent prioritises vector-based semantic search, requiring Azure embeddings and the sqlite-vec extension. As observed in Table 20, it achieved the highest success rates: 100% cosine success for both query types, 88% human success for explicit, and 96% for implicit queries. This demonstrates the substantial value of vector embeddings for semantically complex queries.

Table 20 Vector enhanced agent performance.

Query set	Cosine success (%)	Human Success (%)
Explicit 25	100% (25/25)	88%(22/25)
Implicit 25	100% (25/25)	96% (24/25)

4.3.4 Overview of agentic interaction with relational database

Table 21 summarises the comparative performance of the simple SQL agent, enhanced SQL agent, and vector enhanced SQL agent. There is a clear progression in both accuracy and efficiency from the simple SQL agent to the vector enhanced SQL agent, with vector embeddings notably improving implicit query success rates by 20 percentage points. The enhanced SQL agent achieves the fastest execution time, while the vector enhanced agent balances high accuracy with slightly increased computational overhead. Agreement rates between cosine similarity and human evaluation improve as semantic capabilities increase, indicating better alignment with human judgement. Although engineering complexity rises with capability, the gains in answer quality and reliability justify the investment for production environments. Overall, augmenting SQL agents with semantic and vector-based retrieval substantially enhances their ability to handle complex, context-rich queries, with the vector enhanced SQL agent offering the highest accuracy and robust performance for implicit queries, making it the preferred choice for advanced, real-world applications.

Table 21 Overall Comparison of different types of sql agents.

Feature	Simple SQL agent	Enhanced SQL agent	Vector enhanced SQL agent
Primary Purpose	Basic SQL querying	SQL + Semantic search	SQL + Vector similarity search
Engineering Complexity Level	Basic	Intermediate	Advanced
Uses Full text Index	✗ No	✓ Yes(fallback)	✓ Yes(fallback)
Standard SQL Queries	✓ Yes	✓ Yes	✓ Yes
Need Azure Embeddings	✗ No	⚠ Optional(fallback)	✓ Required (mandatory)
Uses sqlite-vec extension	✗ Not used	⚠ Conditional (tries to load)	✓ Required (mandatory)
Azure AI LLM	gpt-4o	gpt-4o	gpt-4o
Human Evaluation Success Rate (%) in Implicit Queries (25)	76% (19/25)	92% (23/25)	96% (24/25)
Human Evaluation Success Rate (%) in Explicit Queries (25)	84% (21/25)	88% (22/25)	88% (22/25)
Agreement rate (cosine similarity threshold 0.8 vs Human evaluation)	80%	90%	92%
Average Execution time(50 queries) Performance	🟢 Slow(9.48s)	⚡ ⚡ Fastest(6.76s)	⚡ Fast (7.65 s)

4.4 Conclusion from the case studies

This section synthesises the main findings from the three case studies, each exploring a different approach to Retrieval-Augmented Generation (RAG) in manufacturing simulation contexts.

4.4.1 Case study 1: FAISS vector store-based RAG (Local and cloud models)

The first case study demonstrated that both the structure and quality of data, as well as the choice of embedding models, are decisive for retrieval accuracy. Local models using a hybrid retriever (BM25 + FAISS) performed well for explicit queries but struggled with implicit, reasoning-based queries, revealing the limitations of local embeddings and generation models. In contrast, cloud-based models (Azure OpenAI gpt-4o and text-embedding-ada-002) showed marked improvements, especially for implicit queries, with success rates reaching up to 100%. The use of schema-aware prompts and structured plain textual data further enhanced performance, particularly for explicit queries. Notably, simple structured data formats (e.g., plain text with schema) significantly outperformed complex XML-type formats (e.g., Turtle

RDF), underscoring the importance of robust data organisation and prompt engineering for effective RAG applications.

4.4.2 Case study 2: GraphRAG with Neo4j

The second case study evaluated RAG using a knowledge graph approach, modelling the eCatalog as a labelled property graph in Neo4J. Among the retriever architectures tested, the HybridCypherRetriever which combines vector search, full-text search, and graph traversal delivered the highest performance, achieving a 96% success rate for both explicit and implicit queries and perfect agreement between automated and human evaluation. This approach excelled at handling complex, context rich queries by leveraging the relational structure of the knowledge graph. Simpler retrievers, such as pure vector or text to Cypher, were either faster but less context-aware, or more flexible but less reliable. The results also emphasised the importance of careful schema and index design, as these factors directly influenced retrieval quality and responsiveness. Overall, knowledge graphs paired with advanced hybrid retrieval architectures proved particularly effective for domains where relational and contextual information is paramount.

4.4.3 Case study 3: Agentic RAG with SQLite3 database

The third case study focused on LLM-powered agents interacting with a relational SQLite3 database enhanced with full-text and vector indexes. Three agent types were compared: simple SQL agent, enhanced SQL agent (with semantic search), and vector enhanced SQL agent (prioritising vector similarity search). The results revealed a clear progression: the simple SQL agent performed adequately for explicit, fact-based queries but struggled with implicit queries requiring deeper semantic understanding. The enhanced SQL agent improved performance, especially for implicit queries, while the vector enhanced SQL Agent achieved the highest success rates for implicit queries (up to 96%) and the highest agreement with human evaluation. These findings underscore the value of integrating vector-based semantic search into traditional relational querying, especially for complex, natural language queries. However, they also highlight the trade-off between engineering complexity and retrieval quality, with more advanced agents requiring greater development effort but delivering superior results.

5 Discussion and conclusions

The final chapter interprets the results in relation to the research questions and objectives. It discusses the practical implications of deploying LLMs and RAG systems in manufacturing simulation, summarises the main contributions, addresses the limitations of the study, and suggests directions for future research.

5.1 Interpretation of results and summary of findings

This chapter provides a cross-case synthesis of the three retrieval-augmented generation (RAG) approaches evaluated: FAISS vector store-based RAG, GraphRAG with Neo4j, and agentic RAG with Relational Database (SQLite). Using a common QA dataset(25 explicit and 25 implicit queries), each approach results obtained for end-to-end accuracy(retrieval +generation) of system, alignment with human judgement, efficiency, and engineering complexity are compared and presented in Table 22.

Table 22 Key metrics across three case-studies approaches.

Ap- proach/Sys- tem	Best Re- triever/Agent	Human suc- cess rate(%) Explicit	Human suc- cess rate(%) Implicit	Cosine thresh- old-Hu- man agree- ment(%)	Avg.re- sponse Time(s)(50 queries)	Engi- neer- ing Com- plex- ity	Notes/Strenghts
FAISS Vec- tor Store (Local, JSON, Hy- brid)	Hybrid re- triever (BM25+FAISS)	80%	56%	68%	5.74(slow)	Low	Open-source ap- proach without any token cost
FAISS Vec- tor Store (Cloud, JSON, Schema)	Pure Vector + Schema Prompt	100%	100%	100%	2.76(Fast)	Mod- erate	Token cost/API rate limit Top accuracy Speed Scalable
FAISS Vec- tor Store (RDF, Struc- tured)	Pure Vector + Schema Prompt	88%	100%	94%	7.10(Slower)	Mod- erate	Document prepa- ration before em- bedding critical for accuracy
GraphRAG with Neo4j	HybridCypher- Retriever	96%	96%	100%	0.48(Fast- est)	High	Best for context- rich, relational
LLM Agents (SQLite)	Vector En- hanced SQL Agent	88%	96%	92%	7.65(Slow- est)	High	Best for SQL based small da- tabases, seman- tic queries

We present the key comparative insights below:

- Retrieval accuracy:

The highest explicit and implicit human success rates were achieved by the cloud-based FAISS vector store with schema prompts (100% for both), closely followed by the Neo4j HybridCypherRetriever (96% for both). Local hybrid retrieval and SQL agents performed well for explicit queries but lagged for implicit, context-rich queries.

- Cosine-human agreement:
Perfect or near-perfect agreement (100%) was observed for the best-performing cloud and graph-based systems, indicating that automated metrics can reliably reflect human judgement when the retrieval architecture is well designed. Local hybrid retrieval showed much lower agreement (68%), highlighting the limitations of less advanced models and architectures.
- Response time:
The Neo4j HybridCypherRetriever was the fastest (0.48s), making it ideal for real-time, interactive applications. Cloud-based FAISS vector retrieval was also fast (2.76s), while local and SQL-based approaches were slower (5.74-7.65s), reflecting the computational overhead of local processing and semantic search in relational databases.
- Engineering complexity and flexibility:
Simpler vector store approaches (especially with schema-aware prompts) offer a strong balance of accuracy and speed with moderate complexity. Graph based and agentic SQL approaches provide richer context and flexibility but require more sophisticated engineering and infrastructure.
- Data structure sensitivity:
The results across all case studies consistently highlight the pivotal role of data structure in RAG systems. Approaches utilising structured data and schema-aware prompts such as simple structured document with explicit schema or well organised relational tables consistently outperformed those relying on complex formats like RDF Turtle. Notably, even when RDF data was prepared with ontological thinking, retrieval accuracy and efficiency were highly dependent on careful preprocessing and the preservation of structural cues before embedding generation. The structured RDF approach, while capable of high accuracy, was slower and required significant document preparation. These findings underscore that, regardless of the underlying technology, robust data modelling and thoughtful preprocessing are essential prerequisites for achieving optimal retrieval performance in RAG systems.

5.2 Practical implications for manufacturing simulation

When developing RAG systems for manufacturing simulation, developers should adopt a holistic, architecture-aware mindset that goes beyond simply

choosing the most advanced cloud model. Start by evaluating the nature of your queries and the complexity of your domain: for high-stakes, context-rich engineering tasks, prioritise graph-based hybrid retrieval (such as Neo4j HybridCypherRetriever) or schema-aware vector retrieval (example: using cloud FAISS) to ensure accuracy, speed, and robustness. For rapid prototyping or resource-constrained environments, schema-aware vector retrieval offers a strong balance of performance and simplicity, while agentic SQL approaches with vector enhancements are best suited for legacy or SQL-centric systems though they may require more engineering effort and tolerate slower response times.

Regardless of the retrieval architecture, the foundation of success lies in data modelling and preprocessing. Invest time in designing a reusable, ontology-driven data model that captures the relationships and semantics of your manufacturing domain. When using vector retrievers, carefully select and benchmark embedding models, and pay close attention to how you chunk, preprocess, and index your data since the quality of embeddings and the structure of your index storage system directly impact retrieval accuracy and efficiency. For RDF or knowledge graph approaches, ensure that your ontologies are not only theoretically sound but also practically aligned with your retrieval and reasoning needs, and that your data is pre-processed to preserve structural cues before embedding generation.

In summary, effective RAG integration in manufacturing simulation is not just about leveraging the latest AI models, but about orchestrating the right combination of data structure, embedding strategy, index design, and retrieval architecture always with an eye towards scalability, maintainability, and domain-specific requirements.

Final recommendations:

- ✓ For high-stakes, context-rich engineering tasks:
Graph-based hybrid retrieval (Neo4j HybridCypherRetriever) and schema-aware vector retrieval (cloud FAISS) are recommended for their accuracy, speed, and robustness.
- ✓ For rapid prototyping or resource-constrained environments:
Schema-aware vector retrieval offers a strong balance of performance and simplicity.
- ✓ For legacy or SQL-centric environments:
Agentic SQL approaches with vector enhancements provide significant gains, especially for implicit queries, but may require more engineering effort and tolerate slower response times.
- ✓ For open-source or cost-sensitive deployments:
Local hybrid retrieval is viable for explicit queries but may not be suitable for complex, context-rich tasks.

5.3 Answers to research questions

This section addresses the initial research question framed during the thesis.

1)How do different formats of data representation (e.g., JSON, RDF, knowledge graphs, relational table), affect the efficiency and accuracy of information retrieval in RAG systems within the context of manufacturing simulation?

The case studies clearly demonstrate that simple structured data and schema-aware prompts consistently outperform unstructured or flat data formats in RAG systems. When data is well-organised-such as JSON with explicit schema or graph structure with relations retrieval accuracy and efficiency are maximised, as seen in the cloud-based FAISS vector store and Neo4j HybridCypherRetriever results. In contrast, complex but flat formats like RDF Turtle, even when designed with ontological intent, require significant preprocessing and careful chunking to preserve semantic cues; otherwise, retrieval performance suffers.

This finding aligns with the DIKW hierarchy (Figure 5): raw data (e.g., sensor streams, CAD vertices) must be contextualised and structured to become actionable information and knowledge. Only when data is transformed into meaningful information through contextualisation, categorisation, and condensation can RAG systems effectively extract and reason over it. Thus, the journey from data to wisdom in manufacturing simulation is fundamentally enabled by robust data modelling and preprocessing, which serve as the foundation for accurate, context-aware AI retrieval.

2)What is the role of ontologies in improving information retrieval, and what best practices can be established for integrating ontologies with RAG architectures?

Ontologies play a pivotal role in bridging the gap between raw data and actionable knowledge. By providing a formal, shared vocabulary and explicit relationships between manufacturing concepts, ontologies enable RAG systems to interpret queries more accurately, retrieve relevant information from heterogeneous sources, and support context-aware reasoning. The case studies show that knowledge graph-based approaches (e.g., Neo4j HybridCypherRetriever) excel at handling complex, relational queries precisely because they leverage ontological structures. Providing a data schema description in the generation prompt for FAISS retrieval with JSON data demonstrates that clear conceptual definitions significantly enhance results, underscoring the importance of context-specific concept definition when developing ontology-based data models.

Best practices for integrating ontologies with RAG architectures include:

- **Ontology-driven data modelling:** Design data schemas that reflect domain concepts and relationships, enabling semantic search and reasoning.

- Preservation of structural cues: During preprocessing and embedding, maintain the integrity of ontological relationships to support multi-hop and context-rich retrieval.
- Layered knowledge representation: Use ontologies to map data and information to higher-level knowledge and wisdom, as illustrated in the DIKW pyramid, ensuring that LLMs can access not just facts but also the rationale and context behind them.

In essence, ontologies transform information retrieval from simple fact extraction to knowledge-driven reasoning, supporting both novice and expert users in simulation environments.

3) What are the practical benefits and limitations of deploying RAG systems in data from manufacturing simulation environments, particularly regarding retrieval accuracy, system reliability, and engineering complexity? The practical benefits of RAG system in manufacturing simulation data are substantial:

- High retrieval accuracy: Advanced RAG systems, especially those using schema-aware vector retrieval or knowledge graphs, achieve near-perfect alignment with human judgement for both explicit and implicit queries.
- Contextual and relational reasoning: By leveraging structured data and ontologies, these systems can answer not only direct questions but also complex, context-rich queries that require synthesis and inference.
- Support for diverse knowledge types: As shown in Figure 7 the knowledge types and user relation, RAG can surface explicit, procedural, declarative, and even tacit knowledge, supporting users at all levels of expertise.

However, there are limitations:

- Engineering complexity: The most accurate and context-aware systems (e.g., graph-based hybrid retrievers) require significant investment in data modelling, ontology design, and index management.
- Performance trade-offs: While cloud-based and graph-based systems are fast and robust, local or SQL-based approaches may be slower or less effective for complex queries.
- Dependence on data quality: The effectiveness of RAG is fundamentally limited by the quality and structure of the underlying data; poor data modelling or inadequate preprocessing can undermine system performance.

Overall, the deployment of RAG system in manufacturing simulation data is most successful when it is grounded in the principles of the DIKW framework and knowledge engineering. By investing in structured, semantically rich data, robust ontologies, and hybrid retrieval strategies, practitioners can build RAG systems that not only retrieve information but also support

knowledge creation, decision-making, and continuous learning across the full spectrum of simulation tasks.

5.4 Academic and practical contributions

For practitioners in manufacturing simulation and related engineering domains, these findings underscore the necessity of robust data modelling, schema design, and the integration of hybrid or vector-based retrieval methods. Investing in knowledge graphs and ontologies, and adopting schema-aware prompt engineering, can substantially enhance the accuracy and reliability of RAG systems.

From an academic perspective, this study makes a significant contribution by bridging theoretical knowledge engineering concepts specifically the DIKW (Data-Information-Knowledge-Wisdom) hierarchy and the taxonomy of knowledge types with practical applications in manufacturing simulation. The author has mapped these frameworks to the context of simulation software, providing a structured approach that can be adopted across different platforms in the manufacturing simulation community. This synthesis not only advances the theoretical understanding of RAG integration but also offers a replicable, domain-agnostic methodology for future research and development in intelligent simulation environments.

5.5 Limitations of the study and future research

This section outlines the limitations and discusses the future research area.

5.5.1 Limitations of the study

Despite the breadth and depth of this research, several limitations should be acknowledged to contextualise the findings and guide future work:

- Domain and dataset specificity:
The study was conducted using proprietary eCatalog data from manufacturing simulation, which, while highly relevant for industrial applications, may limit the generalisability of the results to other domains or less-structured datasets. The performance of RAG systems and LLM agents may differ when applied to domains with different data characteristics, knowledge structures, or user requirements.
- Model and embedding selection:
The research primarily compared pre-trained, publicly available models (e.g., gpt-4o, text-embedding-ada-002, Llama3, bge-m3) without fine-tuning on domain specific corpora. While this reflects a realistic deployment scenario for many organisations, it may not capture the full potential of LLMs tailored to specific industrial vocabularies or workflows.

- Evaluation metrics and human judgement:
The study relied on a combination of cosine similarity thresholds and human evaluation for assessing retrieval quality. While this dual approach increases reliability, it is still subject to the subjectivity of human judgement and the limitations of automated metrics, especially for nuanced or context-dependent queries.
- Engineering complexity and resource constraints:
The comparative analysis highlighted that the most accurate and context-aware systems (e.g., graph-based hybrid retrievers, vector-enhanced SQL agents) require significant engineering effort, computational resources, and ongoing maintenance. The study did not conduct a detailed cost-benefit analysis or address the operational challenges of scaling these systems in production environments.
- User-centred and longitudinal evaluation:
The research did not include longitudinal studies or direct user feedback from practitioners deploying these systems in live manufacturing simulation projects. As such, the long-term impact on user productivity, learning curves, and trust in AI-augmented tools remains an open question.
- Language and query scope:
All experiments were performed in English and focused on technical queries typical of manufacturing simulation environments. The effectiveness of the evaluated architectures for multi-lingual, colloquial, or highly ambiguous queries remains unexplored.

5.5.2 Directions for future research

Building on these limitations, several avenues for future research are recommended:

- 1) Domain adaptation and fine-tuning:
Investigate the impact of fine-tuning LLMs and embedding models on domain-specific corpora, including the use of transfer learning and continual learning strategies to adapt to evolving manufacturing knowledge and terminology.
- 2) Bridging DIKW and knowledge engineering in practice:
Further develop methodologies that operationalise the DIKW hierarchy and knowledge types (explicit, procedural, declarative, tacit, etc.) within LLM-augmented simulation platforms, ensuring that AI systems not only retrieve information but also support the creation, validation, and transfer of actionable knowledge and wisdom.
- 3) Cost, scalability, and sustainability analysis:

Conduct comprehensive studies on the computational, financial, and environmental costs of deploying advanced RAG systems at scale, including strategies for optimising resource usage and ensuring sustainable AI adoption in manufacturing.

- 4) User-centred design and human-in-the-loop systems:
Incorporate direct user feedback, participatory design, and human-in-the-loop evaluation to better understand the practical needs, trust factors, and usability challenges faced by engineers and simulation practitioners.
- 5) Dynamic knowledge integration and real-time data:
Explore architectures that can dynamically integrate real-time data streams, sensor inputs, and evolving knowledge bases, enabling LLM-powered systems to support adaptive decision-making and continuous improvement in manufacturing simulation.
- 6) Multilingual and cross-domain evaluation:
Extend the evaluation to include multi-lingual datasets, cross-domain scenarios, and more diverse query types, to assess the robustness and adaptability of RAG architectures in global and heterogeneous industrial contexts.

AI Tools used during thesis

For the transparency purposes, author of the thesis declared the use of following AI Tools during the thesis:

Microsoft Copilot: Editing the word document and written text.

GitHub Copilot: Assistant for writing python codes required.

Perplexity: Domain understanding for new topic to author using deep research capabilities.

References

- Alex Garcia. (2025). *GitHub - asg017/sqlite-vec: A vector search SQLite extension that runs anywhere!* <https://github.com/asg017/sqlite-vec>
- Anthropic. (2025). *Models overview - Claude Docs*.
<https://docs.claude.com/en/docs/about-claude/models/overview>
- Arp, Robert., Smith, Barry., & Spear, A. D. . (2015). *Building ontologies with basic formal ontology*. The MIT Press.
- Bang, Y., Cahyawijaya, S., Lee, N., Dai, W., Su, D., Wilie, B., Lovenia, H., Ji, Z., Yu, T., Chung, W., Do, Q. V., Xu, Y., & Fung, P. (2023). *A Multi-task, Multilingual, Multimodal Evaluation of ChatGPT on Reasoning, Hallucination, and Interactivity*. 675–718.
<https://doi.org/10.18653/v1/2023.ijcnlp-main.45>
- Bender, E. M., Gebru, T., McMillan-Major, A., & Shmitchell, S. (2021). On the Dangers of Stochastic Parrots. *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency*, 610–623.
<https://doi.org/10.1145/3442188.3445922>
- Bhaskar, M., & Craswell, N. (2017). *Neural Models for Information Retrieval*.
<https://arxiv.org/pdf/1705.01509>
- Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., Brynjolfsson, E., Buch, S., Card, D., Castellon, R., Chatterji, N., Chen, A., Creel, K., Davis, J. Q., Demszky, D., ... Liang, P. (2022). *On the Opportunities and Risks of Foundation Models*.
- Brin, S., & Page, L. (1998). The anatomy of a large-scale hypertextual Web search engine. *Computer Networks and ISDN Systems*, 30(1–7), 107–117. [https://doi.org/10.1016/S0169-7552\(98\)00110-X](https://doi.org/10.1016/S0169-7552(98)00110-X)
- Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., ... Amodei, D. (2020). *Language Models are Few-Shot Learners*.
- Cao, Q., Samet, A., Zanni-Merk, C., de Bertrand de Beuvron, F., & Reich, C. (2020). Combining chronicle mining and semantics for predictive maintenance in manufacturing processes. *Semantic Web*, 11(6), 927–948.
<https://doi.org/10.3233/SW-200406>
- Cheng, H., Xue, L., Wang, P., Zeng, P., & Yu, H. (2017). Discrete manufacturing ontology development. *2017 IEEE International Conference on Industrial Technology (ICIT)*, 1393–1396.
<https://doi.org/10.1109/ICIT.2017.7915568>
- Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., Schuh, P., Shi, K., Tsvyashchenko, S., Maynez, J., Rao, A., Barnes, P., Tay, Y., Shazeer, N., Prabhakaran, V., ... Fiedel, N. (2022). *PaLM: Scaling Language Modeling with Pathways*.

- Davenport Thomas, & Prusak Laurence. (1998). *Working Knowledge: How Organizations Manage What They Know*.
<https://doi.org/10.1145/348772.348775>
- Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding. *Proceedings of the 2019 Conference of the North*, 4171–4186.
<https://doi.org/10.18653/v1/N19-1423>
- Drobnjakovic, M., Kulvatunyong, B., Ameri, F., Will, C., Smith, B., & Jones, A. (2022). The Industrial Ontologies Foundry (IOF) Core Ontology. *CEUR Workshop Proceedings*, 3240. <https://www.scopus.com/pages/publications/85139845965>
- Emmert-Streib, F. (2023). What Is the Role of AI for Digital Twins? *AI*, 4(3), 721–728. <https://doi.org/10.3390/AI4030038>
- Gage, P. (1994). A new algorithm for data compression. *The C Users Journal Archive*.
- Google DeepMind. (2025a). *Gemini - Google DeepMind*. <https://deepmind.google/models/gemini/>
- Google DeepMind. (2025b). *Gemma - Google DeepMind*. <https://deepmind.google/models/gemma/>
- Guarino, N., Oberle, D., & Staab, S. (2009). What Is an Ontology? In *Handbook on Ontologies* (pp. 1–17). Springer Berlin Heidelberg.
https://doi.org/10.1007/978-3-540-92673-3_0
- Hitzler, Pascal., Krötzsch, Markus., & Rudolph, Sebastian. (2010). *Foundations of Semantic Web technologies*. CRC Press.
- Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8), 1735–1780.
<https://doi.org/10.1162/neco.1997.9.8.1735>
- Hogan, A., Blomqvist, E., Cochez, M., D’amato, C., Melo, G. De, Gutierrez, C., Kirrane, S., Gayo, J. E. L., Navigli, R., Neumaier, S., Ngomo, A.-C. N., Polleres, A., Rashid, S. M., Rula, A., Schmelzeisen, L., Sequeda, J., Staab, S., & Zimmermann, A. (2022). Knowledge Graphs. *ACM Computing Surveys*, 54(4), 1–37. <https://doi.org/10.1145/3447772>
- Huang, L., Yu, W., Ma, W., Zhong, W., Feng, Z., Wang, H., Chen, Q., Peng, W., Feng, X., Qin, B., & Liu, T. (2023). *A Survey on Hallucination in Large Language Models: Principles, Taxonomy, Challenges, and Open Questions*. <https://github.com/>
- Huerga-Pérez, N., Álvarez, R., Ferrero-Guillén, R., Martínez-Gutiérrez, A., & Díez-González, J. (2025). *Optimization of embeddings storage for RAG systems using quantization and dimensionality reduction techniques*.
- IEEE. (2015). *IEEE SA - IEEE 1872-2015*. <https://standards.ieee.org/ieee/1872/5354/>
- Indyk, P., & Motwani, R. (1998). Approximate nearest neighbors. *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing - STOC '98*, 604–613. <https://doi.org/10.1145/276698.276876>
- Joachims, T. (2002). Optimizing search engines using clickthrough data. *Proceedings of the Eighth ACM SIGKDD International Conference on*

- Knowledge Discovery and Data Mining*, 133–142.
<https://doi.org/10.1145/775047.775067>
- Johnson, J., Douze, M., & Jegou, H. (2021). Billion-Scale Similarity Search with GPUs. *IEEE Transactions on Big Data*, 7(3), 535–547.
<https://doi.org/10.1109/TBDATA.2019.2921572>
- Karpukhin, V., Oguz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., & Yih, W. (2020). Dense Passage Retrieval for Open-Domain Question Answering. *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 6769–6781.
<https://doi.org/10.18653/v1/2020.emnlp-main.550>
- Kingma, D. P., & Ba, J. (2017). *Adam: A Method for Stochastic Optimization*.
- Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2017). ImageNet classification with deep convolutional neural networks. *Communications of the ACM*, 60(6), 84–90. <https://doi.org/10.1145/3065386>
- LangChain. (2025a). *Agents - Docs by LangChain*.
<https://docs.langchain.com/oss/python/langchain/agents>
- LangChain. (2025b). *GitHub - langchain-ai/langchain: 🦜🔗 Build context-aware reasoning applications*. <https://github.com/langchain-ai/langchain>
- LangChain. (2025c). *SQLDatabase Toolkit - Docs by LangChain*.
https://docs.langchain.com/oss/python/integrations/tools/sql_database
- Lemaignan, S., Siadat, A., Dantan, J. Y., & Semenenko, A. (2006). MASON: A proposal for an ontology of manufacturing domain. *Proceedings - DIS 2006: IEEE Workshop on Distributed Intelligent Systems - Collective Intelligence and Its Applications*, 2006, 195–200.
<https://doi.org/10.1109/DIS.2006.48>
- Listl, F. G., Fischer, J., & Weyrich, M. (2023). An Architecture for Knowledge Graph based Simulation Support. *IEEE International Conference on Emerging Technologies and Factory Automation, ETFA, 2023-September*. <https://doi.org/10.1109/ETFA54631.2023.10275514>
- Liu, X., Yu, H., Zhang, H., Xu, Y., Lei, X., Lai, H., Gu, Y., Ding, H., Men, K., Yang, K., Zhang, S., Deng, X., Zeng, A., Du, Z., Zhang, C., Shen, S., Zhang, T., Su, Y., Sun, H., ... Tang, J. (2025). *AgentBench: Evaluating LLMs as Agents*.
- Liu, Z., Lin, Y., & Sun, M. (2020). *Representation Learning for Natural Language Processing*. Springer Nature Singapore.
<https://doi.org/10.1007/978-981-15-5573-2>
- Liu, Z., Qiao, A., Neiswanger, W., Wang, H., Tan, B., Tao, T., Li, J., Wang, Y., Sun, S., Pangarkar, O., Fan, R., Gu, Y., Miller, V., Zhuang, Y., He, G., Li, H., Koto, F., Tang, L., Ranjan, N., ... Xing, E. P. (2023). *LLM360: Towards Fully Transparent Open-Source LLMs*.
<https://arxiv.org/pdf/2312.06550>
- Makatura, L., Foshey, M., Wang, B., Hähnlein, F., Ma, P., Deng, B., Tjandrasuwita, M., Spielberg, A., Owens, C. E., Chen, P. Y., Zhao, A., Zhu, A., Norton, W. J., Gu, E., Jacob, J., Li, Y., Schulz, A., & Matusik, W. (2024). Large Language Models for Design and Manufacturing. *An MIT Exploration of Generative AI*.
<https://doi.org/10.21428/E4BAEDD9.745B62FA>

- Malkov, Y. A., & Yashunin, D. A. (2020). Efficient and Robust Approximate Nearest Neighbor Search Using Hierarchical Navigable Small World Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 42(4), 824–836. <https://doi.org/10.1109/TPAMI.2018.2889473>
- McClelland, R. S. (2022). Generative design and digital manufacturing: using AI and robots to build lightweight instrument structures. In A. Padilla-Vivanco, R. B. Johnson, V. N. Mahajan, & S. Thibault (Eds.), *Current Developments in Lens Design and Optical Engineering XXIII* (p. 37). SPIE. <https://doi.org/10.1117/12.2646476>
- Meta Ollama. (2025). *Ollama's documentation - Ollama*. <https://docs.ollama.com/>
- Microsoft. (2025a). *AI Model Catalog | Microsoft Foundry Models*. <https://ai.azure.com/catalog/models/gpt-4o>
- Microsoft. (2025b). *AI Model Catalog | Microsoft Foundry Models*. <https://ai.azure.com/catalog/models/text-embedding-ada-002>
- Microsoft. (2025c). *AI Model Catalog | Microsoft Foundry Models*. <https://ai.azure.com/catalog/models/text-embedding-3-large>
- Microsoft. (2025d). *GitHub - microsoft/semantic-kernel: Integrate cutting-edge LLM technology quickly and easily into your apps*. <https://github.com/microsoft/semantic-kernel>
- Microsoft. (2025e). *SQL Server Management Studio (SSMS) | Microsoft Learn*. <https://learn.microsoft.com/en-us/ssms/sql-server-management-studio-ssms>
- Microsoft. (2025f). *What is Microsoft Foundry? - Microsoft Foundry | Microsoft Learn*. <https://learn.microsoft.com/en-us/azure/ai-foundry/what-is-azure-ai-foundry?view=foundation-classic#microsoft-foundry-portals>
- Nagy, L., Ruppert, T., & Abonyi, J. (2021). Ontology-Based Analysis of Manufacturing Processes: Lessons Learned from the Case Study of Wire Harness Production. *Complexity*, 2021(1). <https://doi.org/10.1155/2021/8603515>
- Nassehi, A., & Urgo, M. (2019). Simulation of Manufacturing Systems. *CIRP Encyclopedia of Production Engineering*, 1570–1573. https://doi.org/10.1007/978-3-662-53120-4_6572
- Naveed, H., Khan, A. U., Qiu, S., Saqib, M., Anwar, S., Usman, M., Akhtar, N., Barnes, N., & Mian, A. (2024). *A Comprehensive Overview of Large Language Models*.
- Neo4j. (2025). *GraphRAG for Python — neo4j-graphrag-python documentation*. <https://neo4j.com/docs/neo4j-graphrag-python/current/>
- Nonaka Ikujiro, & Hirotaka Takeuchi. (1995). *The knowledge-creating company : how Japanese companies create the dynamics of innovation*. Oxford University Press.
- OPENAI. (2025). *Models - OpenAI API*. <https://platform.openai.com/docs/models>
- Ostendorff, M., Suarez, P. O., Lage, L. F., & Rehm, G. (2024). *LLM-Datasets: An Open Framework for Pretraining Datasets of Large Language Models*. <https://commoncrawl.org>

- Prestes, E., Carbonera, J. L., Rama Fiorini, S., Vitor, V. A., Abel, M., Madhavan, R., Locoro, A., Goncalves, P., E. Barreto, M., Habib, M., Chibani, A., Gérard, S., Amirat, Y., & Schlenoff, C. (2013). Towards a core ontology for robotics and automation. *Robotics and Autonomous Systems*, 61(11), 1193–1204. <https://doi.org/10.1016/J.RO-BOT.2013.04.005>
- Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., & Liu, P. J. (2023). *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*.
- Ramírez-Durán, V. J., Berges, I., & Illarramendi, A. (2020). ExtrOnt: An ontology for describing a type of manufacturing machine for Industry 4.0 systems. *Semantic Web*, 11(6), 887–909. <https://doi.org/10.3233/SW-200376>
- Robertson, S. E., & Jones, K. S. (1976). Relevance weighting of search terms. *Journal of the American Society for Information Science*, 27(3), 129–146. <https://doi.org/10.1002/asi.4630270302>
- Rowley, J. (2007). The wisdom hierarchy: representations of the DIKW hierarchy. *Journal of Information Science*, 33(2), 163–180. <https://doi.org/10.1177/0165551506070706>
- Russell Lincoln Ackoff. (1989). From data to wisdom. *Journal of Applied Systems Analysis*, 16, 3–9.
- Sachdeva, N., Coleman, B., Kang, W.-C., Ni, J., Hong, L., Chi, E. H., Caverlee, J., McAuley, J., & Cheng, D. Z. (2024). *How to Train Data-Efficient LLMs*.
- Salton, G., Wong, A., & Yang, C. S. (1975). A vector space model for automatic indexing. *Communications of the ACM*, 18(11), 613–620. <https://doi.org/10.1145/361219.361220>
- Sapel, P., Molinas Comet, L., Dimitriadis, I., Hopmann, C., & Decker, S. (2024). A review and classification of manufacturing ontologies. *Journal of Intelligent Manufacturing*, 1–25. <https://doi.org/10.1007/S10845-024-02425-Z/TABLES/6>
- Schäfer, P. M., Steinmetz, F., Schneyer, S., Bachmann, T., Eiband, T., Lay, F. S., Padalkar, A., Sürig, C., Stulp, F., & Nottensteiner, K. (2021). Flexible Robotic Assembly Based on Ontological Representation of Tasks, Skills, and Resources. *Proceedings of the Eighteenth International Conference on Principles of Knowledge Representation and Reasoning*, 702–706. <https://doi.org/10.24963/kr.2021/73>
- Shaikh, O., Zhang, H., Held, W., Bernstein, M., & Yang, D. (2023). *On Second Thought, Let's Not Think Step by Step! Bias and Toxicity in Zero-Shot Reasoning*. <https://github.com/SALT-NLP/>
- Smith, S., Patwary, M., Norick, B., LeGresley, P., Rajbhandari, S., Casper, J., Liu, Z., Prabhumoye, S., Zerveas, G., Korthikanti, V., Zhang, E., Child, R., Aminabadi, R. Y., Bernauer, J., Song, X., Shoeybi, M., He, Y., Houston, M., Tiwary, S., & Catanzaro, B. (2022). *Using DeepSpeed and Megatron to Train Megatron-Turing NLG 530B, A Large-Scale Generative Language Model*.
- SQLite. (2025a). *SQLite FTS5 Extension*. <https://sqlite.org/fts5.html>

- SQLite. (2025b). *SQLite Home Page*. <https://sqlite.org/index.html>
- Sun, J., Xu, C., Tang, L., Wang, S., Lin, C., Gong, Y., Ni, L. M., Shum, H.-Y., & Guo, J. (2024). *Think-on-Graph: Deep and Responsible Reasoning of Large Language Model on Knowledge Graph*.
- Tanyildiz, D., Ayvaz, S., & Amasyali, M. F. (2024). Enhancing Retrieval-Augmented Generation Accuracy with Dynamic Chunking and Optimized Vector Search. *Orclever Proceedings of Research and Development*, 5(1), 215–225. <https://doi.org/10.56038/oprd.v5i1.516>
- Tinsel, E.-F., Lechler, A., Riedel, O., & Verl, A. (2024). Concept of an Initial Requirements-Driven Factory Layout Planning and Synthetic Expert Verification for Industrial Simulation Based on LLM. *2024 IEEE 22nd International Conference on Industrial Informatics (INDIN)*, 1–6. <https://doi.org/10.1109/INDIN58382.2024.10774366>
- Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., & Lample, G. (2023). *LLaMA: Open and Efficient Foundation Language Models*.
- Umbrico, A., Orlandini, A., & Cesta, A. (2020). An Ontology for Human-Robot Collaboration. *Procedia CIRP*, 93, 1097–1102. <https://doi.org/10.1016/J.PROCIR.2020.04.045>
- Usman, Z., Young, R. I. M., Chungoora, N., Palmer, C., Case, K., & Harding, J. A. (2013). Towards a formal manufacturing reference ontology. *International Journal of Production Research*, 51(22), 6553–6572. <https://doi.org/10.1080/00207543.2013.801570>
- Valmeekam, K., Stechly, K., & Kambhampati, S. (2024). *LLMS STILL CAN'T PLAN; CAN LRMS? A PRELIMINARY EVALUATION OF OPENAI'S O1 ON PLANBENCH*.
- Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). *Attention Is All You Need*. <https://arxiv.org/abs/1706.03762>
- Visual Components. (2025). *Fastest way from concept to reality - Visual Components*. <https://www.visualcomponents.com/>
- Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., & Zhou, D. (2023). *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*.
- Werheid, J., Melnychuk, O., Zhou, H., Huber, M., Rippe, C., Joosten, D., Keskin, Z., Wittstamm, M., Subramani, S., Drescher, B., Göppert, A., Abdelrazeq, A., & Schmitt, R. H. (2024). *Designing an LLM-Based Copilot for Manufacturing Equipment Selection*.
- Xia, Y., Dittler, D., Jazdi, N., Chen, H., & Weyrich, M. (2024). *LLM experiments with simulation: Large Language Model Multi-Agent System for Simulation Model Parametrization in Digital Twins*. <https://arxiv.org/abs/2405.18092v2>
- Xiao, G., Lin, J., Seznec, M., Wu, H., Demouth, J., & Han, S. (2023). *SmoothQuant: Accurate and Efficient Post-Training Quantization for Large Language Models*. <https://github.com/mit-han-lab/smoothquant>

- Yang, C., Hu, X., Lu, J., & Cheng, Q. (2025). *Large Language Model-Enable Knowledge Exploration for Automotive Assembly Process Using Ontology*. <https://doi.org/10.2139/SSRN.5149347>
- Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). *ReAct: Synergizing Reasoning and Acting in Language Models*.
- Zhang, Z.-X., Wen, Y.-B., Lyu, H.-Q., Liu, C., Zhang, R., Li, X.-Q., Wang, C., Du, Z.-D., Guo, Q., Li, L., Zhou, X.-H., & Chen, Y.-J. (2025). AI Computing Systems for Large Language Models Training. *Journal of Computer Science and Technology*, 40(1), 6–41. <https://doi.org/10.1007/s11390-024-4178-1>
- Zhao, P., Zhang, H., Yu, Q., Wang, Z., Geng, Y., Fu, F., Yang, L., Zhang, W., Jiang, J., & Cui, B. (2024). *Retrieval-Augmented Generation for AI-Generated Content: A Survey*. <https://github.com/PKU-DAIR/RAG-Survey>.
- Zhao, S., Yang, Y., Wang, Z., He, Z., Qiu, L. K., & Qiu, L. (2024). *Retrieval Augmented Generation (RAG) and Beyond: A Comprehensive Survey on How to Make your LLMs use External Data More Wisely*. <https://arxiv.org/abs/2409.14924v1>