Universiteit
Leiden
The Netherlands

Bachelor Computer Science & Economics

A Domain-Specific Intelligent Agent Leveraging RAG
for Legacy-to-Modern Data Migration

Talha Çelik

Supervisors:
Zhaochun Ren, Zhao Jujia - Leiden University

BACHELOR THESIS

Leiden Institute of Advanced Computer Science (LIACS)
www.liacs.leidenuniv.nl

03/07/2026

Abstract

Legacy-to-modern data migration is hindered by heterogeneous source artifacts and the limitations of Large Language Models (LLMs) that prevent their reliable use on the task, including stale knowledge, hallucination, finite context, and weak coverage of specialized enterprise vocabularies. This thesis investigates whether Retrieval-Augmented Generation embedded in an agentic workflow can mitigate these limitations and produce accurate and traceable schema and protocol mappings with minimal human intervention.

A domain-specific Schema-Mapping Agent is built on Microsoft Azure AI Foundry with GPT-4.1, a knowledge base of legacy schemas, target profiles, and mapping rules, and an iterative loop. It was compared against an LLM-only baseline and a Traditional RAG baseline across five use cases spanning schema mapping, industrial validation, cross-domain healthcare conversion, orchestration translation, and open-ended design.

The agent reliably outperformed the LLM-only baseline whenever the task required discipline that the model did not bring on its own. This is most decisive for orchestration migration, where preserving source provenance produces the largest separation observed in this study. For tasks within the model's parametric capability, the retrieval and agent layers add little that the metrics can detect. Retrieval-augmented agentic reasoning is therefore most useful for migration tasks that require discipline beyond source-target translation. It produces accurate and auditable outputs where the model alone falls short.

Contents

1 Introduction

Despite decades of investment in digital transformation, approximately 75% of enterprise IT infrastructure may depend on systems built before cloud computing, real-time data processing, and modern integration standards [Ann25]. Organizations spend more than half of their IT budgets modernizing these legacy systems [Ann25]; however, replacing them remains a formidable challenge because the data they hold exhibit high variability in terms of their type, format, and origin [Wan17]. This variability is not an underestimated issue because heterogeneous data sources differ along dimensions: syntactic dimensions involve different serialization languages and structures, conceptual dimensions involve business entities modelled differently across the system, and terminological dimensions involve different names referring to the same real concept [Wan17]. Each of these mismatches must be corrected before data is moved from a legacy system to a modern target without any information loss [Kou25]. This has been seen as the highest risk of any modernization activity [PE23, OOJ+23]. Understanding how Artificial Intelligence (AI) tools can help with the legacy of modern target transformation is the central motivation of this study.

Large Language Models (LLMs) have transformed natural language processing through strong performance in generation, summarization, and question answering [VSP+17, CHGD24], and have begun to be deployed across finance, healthcare, legal, and education sectors [CHGD24]. However, their reliable deployment in enterprise settings, such as legacy-to-modern, is constrained by four limitations. *Stale knowledge*, since training cut-offs leave the model unaware of specific schemas and recent target conventions [MAR25]. *Hallucination*, where the outputs are confident but unfaithful or unverifiable [JLF+23], especially when an incorrect schema mapping propagates through an entire pipeline. *Limited context capacity*, which constrains how much schema documentation and mapping history the model can reason over at once [CTS+24]. *Weak domain-specific knowledge*, since general-purpose training corpora under-represent the specialised vocabularies of enterprise integration [CHGD24]. Retrieval-Augmented Generation (RAG) was introduced precisely to mitigate these limitations by giving the model access to external knowledge sources at inference time, grounding generation in evidence the model would not otherwise see [LPP+20, SPC+21, CTS+24]. These capabilities are important for legacy system modernization because the gaps RAG closes align with the gaps legacy migration exposes. Legacy system modernization need the domain-specific knowledge that is rarely present in public training corpus, and this is not covered by general LLM knowledge [OOJ+23, PE23]. Migration also demands traceability between the source and target, which a hallucination-prone model cannot provide without external grounding [PE23].

Modernization a strategic priority for organizations addressing technical debt, cybersecurity exposure, and competitive disadvantage [Ann25, OOJ+23], and retrieval-augmented approaches are matched with these necessities. However, the standard formulation of RAG operates as a linear single-pass workflow with no support for error correction or multi-step reasoning, which becomes brittle in knowledge-intensive tasks where the right context cannot be retrieved in one shot [MNY+26]. To overcome these structural limits, RAG is reframed as a sequential decision-making process, giving rise to Agentic RAG architectures in which the LLM acts as an autonomous agent that iteratively plans, retrieves, reasons, and verifies [MNY+26].

This study aims to investigate the role of RAG-based intelligent agents in supporting legacy-to-

modern system migration and integration tasks. Specifically, this thesis examines how agent-based architectures can be used to interpret and transform heterogeneous data formats and support schema and protocol mapping processes within enterprise workflows.
This research was conducted in the context of a graduation internship at Capgemini. Capgemini is a company specialising in consulting, technology services, and digital transformation. The internship takes place within the Integration cluster, where interoperability, migration, and transformation of legacy systems into modern environments are highly prominent. This setting provides a valuable context for investigating how intelligent agents and AI-supported approaches may help in migration and integration tasks involving heterogeneous data formats, schema transformation, and protocol mapping.

Research Questions The main research question is as follows:
RQ *How can intelligent agents be designed to autonomously interpret and transform heterogeneous data formats during legacy-to-modern system migration, leveraging Retrieval-Augmented Generation to map disparate schemas and protocols with minimal human intervention?*

There are two additional sub-questions.
SQ 1- *Which stages of legacy-to-modern migration can be supported by intelligent agents when dealing with heterogeneous data formats? What are the roles of intelligent agents?*
SQ 2- *How can an intelligent agent leverage iterative retrieval, reasoning, and transformation mechanisms to map disparate schemas and protocols accurately?*

2 Background & Related Work

This section surveys the body of literature that underpins the present research. It begins by examining legacy systems and the challenges they pose for modernization before turning to the technical complexities of heterogeneous data integration and schema mapping. It then introduces the landscape of AI Agents and Agentic architecture. The paper concludes by tracing how these threads converge in recent work on LLM-driven schema matching and data pipeline automation.

2.1 Legacy Systems

Legacy systems are recognized as the backbone of organizational information flow, acting as the primary means of consolidating and managing business data [BLW+97]. Regardless of their importance, these systems pose significant challenges for organizations that depend on them. Many legacy systems operate on outdated hardware that is slow and costly to maintain, whereas software maintenance itself often entails high expenses [BLW+97]. As organizations generate and process increasingly large volumes of data, legacy systems struggle to effectively support dynamic and evolving workloads [OOJ+23]. In addition, diagnosing and resolving faults are both time-consuming and costly owing to insufficient documentation and limited understanding of the internal structures of the systems [BLW+97].
According to Althani et al. [AK17], legacy systems can result in lost business opportunities. Integration initiatives are further hindered by the absence of well-defined interfaces, and extending or enhancing legacy systems is frequently difficult, if not infeasible [BLW+97]. Legacy systems

cannot directly utilize the most recent advancements in hardware, such as multicore architecture or the most current software paradigms [AK17]. As new technologies arise to meet the expansion and diversity of users' expectations, service-oriented systems have been growing quickly in recent years [AK17].

2.2 Legacy-to-Modern Migration

To address hardware limitations and the risk of lost business opportunities, organizations must launch legacy-to-modern migration initiatives. This process involves the transition of outdated software and infrastructure into modern environments by upgrading, completely changing the operational environment, or redesigning the existing architecture [AK17].

The modernization of legacy applications has moved beyond being a purely technical upgrade and is now recognized as a strategic business priority. This is largely driven by the need to address technical debt, enhance cybersecurity resilience, reduce operational costs, and improve organizational flexibility [OOJ+23]. Simultaneously, it creates a foundation for adopting scalable and modern paradigms, such as cloud computing, analytics, AI, and microservice architectures [FAA25, PE23, OOJ+23].

Because replacing a functioning legacy system incurs substantial financial costs and operational risks, organizations must carefully assess and choose a modernization strategy that is consistent with their specific business values, risk tolerance, and system quality attributes [AK17]. These transformation strategies exist on a spectrum, ranging from minimal codebase alterations to comprehensive system overhauls [PE23].

Transitioning from legacy systems to decentralized modern infrastructures requires a high level of interoperability, which is often challenged by misaligned heterogeneous data formats, opaque data layers, and outdated communication protocols [PE23]. These integration struggles highlight the necessity of robust data transformation and schema mapping approaches to avoid data corruption, information loss, and functional failures during the migration process.

2.3 Data Mapping, Schema Mapping, and Heterogeneous Data

The legacy-to-modern transformation requires the processing of massive amounts of heterogeneous data, which can be defined as data that exhibit high variability in type, format, and origin and are often characterized by structural inconsistencies, high redundancy, and varying levels of data quality [Wan17]. In modern enterprise environments, managing this heterogeneity inherently entails dealing with structured, semi-structured, and completely unstructured data simultaneously, as these disparate representation forms are frequently interconnected but represented inconsistently [Wan17, SAKS+26]. Structured data is highly normalized and stored in traditional relational tables organized by rows, columns, and relational keys [SAKS+26]. Semi-structured data, such as XML documents, JSON files, or NoSQL databases, do not conform to a strict relational database schema but contain hierarchical or organizational properties that facilitate analysis [SAKS+26]. Most enterprise information comprises unstructured data; the lack of organizational features makes it challenging to effectively utilize elements such as free-form text, system logs, emails, and multimedia content. [SAKS+26, SM21].

Legacy systems have disparate data structures that lack modern standardization, making migration to decentralized environments a significant challenge [SAKS+26]. Different data sources can be

harmonized using data integration workflows, such as Extract, Transform, Load (ETL) or Extract, Load, Transform (ELT) pipelines. These pipelines extract data from databases and transform them to meet the structural requirements of modern target systems [SM21].
The success of these pipelines fundamentally depends on the accurate alignment of the source and target data models [RB01]. This alignment is called schema matching, where the goal is to identify semantic correspondences between the elements of different schemas so that equivalent real-world concepts are accurately mapped [RB01]. Schema mapping may become challenging in legacy system modernization because of incomplete legacy documentation, ambiguous attribute naming, domain-specific abbreviations, and restrictions on the use of actual data values [PVN+25]. In this sensitive enterprise environment, the use of actual data is not possible; therefore, schema mapping is heavily based on manual, instance-free, name-based schema matching [PVN+25]. Because manual schema matching is time-consuming, prone to errors, and labor-intensive, interest in automatic schema matching has increased [PVN+25]. To overcome these obstacles, recent research has leveraged LLMs to automate the semantic interpretation and matching of heterogeneous legacy data [PVN+25, SMOB25].
Fine-tuned LLMs have shown huge potential in assisting with labor-intensive manual data engineering by identifying semantic mappings and providing solutions for automation [SMOB25]. Furthermore, as data grows in volume and complexity, conventional ETL workflows struggle with limited scalability, resource-intensive maintenance, and reactive problem-solving. To tackle these structural vulnerabilities, Chakraborty et al. [Cha25] proposed integrating AI agents into data architectures to allow organizations to develop self-repairing data pipelines that can autonomously validate schemas, dynamically map data, and proactively address errors.

2.4 Large Language Models

A Large Language Model is a neural network-based model trained on large-scale text corpora to model the probability distribution over sequences of tokens [CHGD24]. Specifically, an autoregressive LLM learns to predict each token conditioned on all preceding tokens in the sequence, effectively modeling the joint probability of the entire sequence as a product of conditional probabilities [BMR+20, AAA+23].
In terms of architecture, LLMs are predominantly built on the Transformer Framework, which relies on self-attention mechanisms to capture dependencies between tokens regardless of their distance in the sequence [VSP+17, CGeA24]. Three principal architectural variants exist: encoder-only models, suited for classification and comprehension tasks; decoder-only models, designed for generative tasks such as text completion; and encoder-decoder models, which combine both for tasks requiring understanding and generation [CGeA24].
LLMs can be adapted to specific downstream tasks using two primary mechanisms. The first is fine-tuning, which involves adjusting the model parameters on task-specific data and typically yields stronger task performance at the cost of additional computational resources. The second is In-Context Learning (ICL), which conditions the model on a small number of input-output examples simultaneously without updating any parameters, allowing the model to generalize to new tasks through prompt design alone [BMR+20, CGeA24].

2.4.1 Limitations of LLMs

Despite their remarkable advantages, LLMs exhibit several fundamental limitations that constrain their reliable deployment in enterprises that require extensive knowledge [CHGD24].
The first structural limitation concerns the currency of knowledge. LLMs are static artifacts trained on fixed data snapshots, which means that their internal knowledge reflects the state of the world at a particular point in time [MAR25]. Factual knowledge is dynamic, continually evolving as new information emerges and older data becomes outdated. Therefore, a reliable knowledge system must ensure that its information remains accurate and up to date [MAR25].
The second limitation is consistency, so even if the model has the right knowledge, it may have different outputs when prompts are varied or when new information is introduced that conflicts with its internal representations [MAR25]. A knowledge base that fails to fulfil this requirement risks misleading users and leading to incorrect conclusions and decisions [MAR25].
The third limitation is hallucination, namely the tendency of LLMs to generate fluent, confident-sounding text that is factually incorrect or unverifiable [JLF+23]. Two distinct forms are identified in the literature: intrinsic hallucinations, where the generated output contradicts the source material, and extrinsic hallucinations, where the output contains content that cannot be verified against any source and may be either correct or incorrect [JLF+23]. In critical circumstances, such as schema mapping during legacy migration, where incorrect correspondences can propagate errors throughout an entire data pipeline, this limitation is particularly consequential.
The fourth limitation relates to the context length. LLMs have a finite context window, which constrains their ability to process and reason with extended inputs [CTS+24]. In organizational data integration tasks, where relevant schema documentation, data dictionaries, and mapping history may cover a large amount of text, this directly limits the model's effectiveness.
Finally, LLMs often struggle with domain-specific knowledge, particularly in specialized domains that are not properly represented in general-purpose training datasets [CHGD24]. This limitation makes fine-tuning or external knowledge augmentation necessary for reliable deployment in enterprise settings [Sar22].

2.4.2 Artificial Intelligence Agents

LLMs only perform prompt-based outputs and do not follow autonomous tasks or self-reasoning [SRK25]. The necessity of handling dynamic tasks, maintaining continuity, or performing multi-step planning has led to the development of Artificial Intelligence Agents [SRK25]. AI Agents are single-entity systems empowered with adaptability for environments, making real-time decisions, and autonomously using external tools to complete specific tasks with minimal human intervention [SRK25, Kri25].
Having multiple specialized AI Agents that engage in communication, dynamically break down complex tasks, and share memory states to accomplish high-level goals signifies Agentic AI [SRK25].

2.5 Retrieval-Augmented Generation

2.5.1 Traditional RAG

Retrieval-Augmented Generation (RAG) was introduced to address a core limitation of parametric language models, which is their inability to easily update their knowledge or provide transparent

evidence for their outputs or decisions [LPP+20]. Rather than relying on the knowledge encoded in the model parameters during training, RAG pairs a parametric generator with an external memory component [LPP+20]. This allows the model to condition its outputs on the externally retrieved evidence at inference time and ensure that the outputs are accurate and traceable [LPP+20].
At its core, a traditional RAG system consists of two components: a retriever and a generator [LPP+20]. The retriever component is mainly responsible for taking a user query and searching an external knowledge source, such as a dense vector index, to identify the most relevant passages [CTS+24]. Not only finding the most relevant information, but also passing the relevant information to the generator to produce a grounded response is the task of the retriever [CTS+24]. The second component, the generator, conditions its output simultaneously on the user's initial query and the specific details from the retrieved text; the generator can produce a carefully grounded response [CTS+24].
By generating responses from external knowledge, RAG reduces the tendency of LLMs to hallucinate contents [LPP+20]. Traditional RAG operates on a static, single-pass control flow in which the retriever fetches a fixed set of passages and the generator synthesizes a response without any opportunity for feedback, evaluation, or revision. [MNY+26]. The inflexibility of this design becomes brittle when dealing with knowledge-intensive or multi-hop scenarios, where gathering full context from a single retrieval is mostly not enough [MNY+26, NB25]. This is because it is a struggle to refine queries iteratively, correct intermediate errors, handle multi-step reasoning, or adapt to heterogeneous data sources [NB25].

2.5.2 Agentic RAG

These limitations give rise to Agentic RAG architectures by embedding autonomous AI Agents into the RAG pipeline [SEK+25]. In this setting, an agent is a computational entity that senses its environment, makes decisions independently, and acts to accomplish particular objectives [NB25]. Different from traditional RAG components, agents have the ability to adapt the concept, use extra tools, reason over the different steps that they take, act target-driven, and control tasks in multiple stages [NB25]. These abilities cover six main functions: perception, understanding of input requests, goals, and contextual clues; reasoning, splitting complex pieces into smaller ones, and finding the most effective actions to take next; action, performing retrieval, tool invocation, or generation; reflection, assessing intermediate results, and modifying approaches accordingly; adaptation, learning from previous outcomes; and collaboration, cooperation with other agents in multi-agent environments [NB25].
Agents can take different roles within different systems [NB25]. For instance, a Planner is for breaking down queries into feasible steps, a Retriever Agent performs adaptive or multi-hop retrieval, a QA Agent generates fact-grounded answers, a Tool Agent executes API's or structured tools, a Verifier checks factual consistency, and a Memory Agent keeps memory contextual for retrieval and generation [NB25].
These agents apply agentic reflective principles, such as reflection, planning, tool use, and collaboration among multiple agents, to control all retrieval processes and work iteratively within different workflows [SEK+25]. Working with this Agentic architecture provides flexibility and scalability for multi-domain tasks, contextual awareness, and high accuracy [SEK+25].
In this study, where schema mapping tasks involve incomplete documentation, ambiguous attribute naming, and multi-step reasoning over heterogeneous legacy data, this architectural paradigm

provides the basis for the proposed intelligent agent design.

2.6 Evolving Approaches to Schema Matching and Data Integration

Results have been obtained from the use of LLMs for schema matching and integration automation in enterprise environments [PVN+24]. At a fundamental level, LLMs serve as the core reasoning tool. Parciak et al. have demonstrated that using only item names and descriptions, pre-trained LLMs can start the process of schema mapping, thereby reducing the need for manual verification work without requiring access to raw data samples [PVN+24].
While RAG-based systems resolve the efficiency and data privacy issues of web-searching agents, they may operate as isolated tools and encounter inherent limitations regarding multi-system coordination, zero-trust governance, and security matters [Ven26].
For further modernization of legacy systems with multi-agent designs, Venkiteela introduced the Enterprise Agentic Architecture Framework (EAAF) to position AI Agents as first-class integration units capable of operating across hybrid cloud environments and Integration Platform as a Service (iPaaS) solutions such as SAP BTP and Boomi [Ven26]. These agents can operate autonomously and perform semantic data mapping and dynamic transformations to solve schema mismatches and data anomalies across heterogeneous systems and recover complex enterprise systems [Ven26]. Another utilization of these agents is that they interact with API gateways and iPaaS to autonomously orchestrate complicated integration workflows and self-heal system failures across legacy and modern platforms [Ven26].

3 Methodology

The goal of this study was to design an Agentic RAG system capable of autonomously interpreting legacy enterprise integration artifacts and producing field-level schema mappings to modern target formats. This chapter represents the conceptual framework underlying this system, with the details of the mechanism and the baseline configurations.

3.1 Agentic RAG Framework

The agent is designed around a five-stage workflow: plan, retrieve, transform, verify, and correct. This design extends the single-pass Traditional RAG architecture described in Section 2.5.1 by instructing the agent to evaluate its own output against the retrieved target conventions before returning a final result, following the Agentic RAG paradigm described in Section 2.5.2.

- **PLAN** — the agent interprets the source artifact (e.g., an XSD schema or WSDL contract) and identifies which categories of knowledge are required to complete the mapping: the structure of the legacy source, the conventions of the modern target, and the transformation rules connecting the two.
- **RETRIEVE** — the agent formulates targeted subqueries for each knowledge category identified during planning and retrieves the relevant content from the knowledge base.
- **TRANSFORM** — the agent applies the retrieved schema definitions, target conventions, and mapping rules to produce a candidate output artifact in the target format.

- **VERIFY** — the candidate output is checked against the retrieved target-format conventions for structural and semantic consistency, including field types, required fields, and naming conventions.
- **CORRECT** — inconsistencies identified during verification are resolved by revisiting the relevant retrieved knowledge and regenerating the affected parts of the output.

3.2 Knowledge Base Architecture

The framework separates domain knowledge into three conceptually distinct categories, reflecting the three information types a schema mapping task requires: what the source contains, what the target expects, and a definition of how those two should correspond.

- **Legacy schema knowledge** — structural information about the source artifacts, such as element names, types, nesting, and cardinality.
- **Target profile knowledge** — structural and stylistic conventions of the destination format.
- **Mapping rule knowledge** — explicit field-level and protocol-level correspondence rules between source and target, including type conversion patterns and naming conventions.

This separation of the knowledge base allows the RETRIEVE step to query only the relevant knowledge rather than searching an undifferentiated corpus. The reference material in each category is stored as structured natural-language descriptions rather than as raw technical files, because embedding-based similarity search operates over natural-language tokens; a prose description of an XSD structure is more retrievable than the raw XML artifact itself. This design reflects the instance-free, name-based schema-matching approach in which the agent relies on schema definitions and structured descriptions rather than raw data samples [PVN+25].

3.3 Retrieval Mechanism

Rather than retrieving a single fixed set of passages for the entire task, as in Traditional RAG, retrieval in this framework is performed by an agentic retrieval engine that decomposes each retrieval request into targeted subqueries, routes each subquery to the relevant knowledge category, executes them in parallel, and merges the results before they are passed to the TRANSFORM stage. The decomposition and routing are performed by the retrieval engine itself rather than by the agent's instruction set: the engine uses a reasoning model to formulate the subqueries and select the knowledge sources to query. This behaviour is a documented property of the engine and is externally observable, as each retrieval response includes an activity record listing the subqueries executed, the knowledge sources queried, and the results returned.

3.4 Baseline Configurations

To isolate the contributions of retrieval and agentic reasoning, three configurations of the same underlying language model are compared.

- **B0 (LLM-only)** — the model receives a minimal task prompt together with the source artifacts. B0 has no retrieval and no agentic loop; it captures the raw parametric capability of the language model on the migration task.
- **B1 (Traditional RAG)** — the model receives the same task prompt as B0, augmented with the relevant mapping rules embedded statically in the prompt, and produces its output in a single pass .
- **B2 (Agentic RAG)** — the full PLAN → RETRIEVE → TRANSFORM → VERIFY → CORRECT workflow described in Section 3.1, with dynamic runtime access to the structured knowledge base.

All three configurations share the same underlying language model, sampling temperature, and maximum response length; the only differences are the retrieval mechanism and the presence or absence of the agentic workflow. This isolates the two variables under investigation: the effect of retrieval (B0 versus B1) and the effect of agentic iteration on top of retrieval (B1 versus B2). The shared model and parameter settings are specified in Section 4.

3.5 Data Description

This study draws on five distinct data sources, reflecting both industrial origin and academic standardisation. The starting data for the study were provided by Capgemini and consisted of legacy Oracle SOA Suite 12c artifacts. This dataset does not contain real-life data but rather examples of the schemas and service contracts. The Capgemini dataset forms the first leg of the evaluation and provides an industrial validity ground for the study. The first academic corpus is the Oracle SOA Suite handbook reference repository[1]. This source was selected because it shares the same dataset structure as the Capgemini-provided artifact. Six test cases were extracted from this source, and each test case consisted of an XSD schema, a WSDL service contract, an example XML instance, and a gold-standard JSON representing the equivalent modern Boomi REST artifact. These six cases constitute a primary quantitative benchmark.

The second academic corpus is the HL7 FHIR R4 specification examples, which are the official reference examples published as part of the FHIR healthcare interoperability standard. We used this dataset to increase the amount of data in our experiment. Ten test cases were selected, one per major resource type: Patient, Observation, Condition, MedicationRequest, Encounter, AllergyIntolerance, Procedure, Organization, Practitioner, and DiagnosticReport. Each test case is provided in both FHIR XML and FHIR JSON formats, and the JSON form serves directly as the gold standard.

The third academic source provided two further migration tasks, both drawn from the SOA Suite handbook. A synchronous BPEL process (`QuickAndSlowProcess`, chapter 9 of the handbook) was selected as the input, with an AWS Step Functions ASL JSON state machine prepared as the gold standard. From the same corpus, the `AsynchKelvinToFahrenheitProcessor` asynchronous SOAP service (chapter 23) was selected as another input, with three alternative modern REST design specifications (webhook, polling, and server-sent events) evaluated qualitatively against a six-dimensional rubric.

[1] https://github.com/lucasjellema/soasuitehandbook

3.6 Data Formats

Legacy artifacts in this study exhibit the three classical forms of heterogeneity that arise in enterprise data integration: schema-level, semantic, and instance-level heterogeneity [Kou25]. These properties recur across the Capgemini dataset and the SOA Suite handbook corpus and motivate the design of the agent.

Schema-level heterogeneity appears in the form of classical XML following XSD schemas. Identifiers are encoded as XML attributes (for example, `@number`) for which no direct equivalent exists in JSON. Repeated elements such as `<row>` entries within an order must be reshaped into JSON arrays. The structure of the legacy artifacts is captured in their XSD definitions, which serve as the primary schema reference during transformation.

Semantic heterogeneity is remarkable in attribute naming. The attribute `@number` functions as an order identifier, but the name does not convey these semantics. Mapping this attribute to a target field such as `orderId` therefore requires semantic understanding beyond syntactic equivalence. This is precisely the kind of task on which LLM-based, name-based, instance-free schema matching has demonstrated value [PVN+25].

Instance-level heterogeneity appears in primitive value formats. Numeric values such as amount and price are string-typed in XSD and, in the Capgemini dataset, formatted with a comma as the decimal separator following Dutch localisation (for example, `3499,00`). The modern JSON target requires real numeric types and the international decimal point (`3499.00`). FHIR XML exhibits an analogous instance-level concern: every primitive is serialised as an attribute string (`value="true"`, `value="1"`), and correct conversion to FHIR JSON requires more certain boolean and number types.

The target modern format conventions used across the use cases include flat JSON fields, typed numerical values, and JSON arrays for repeated entities. Depending on the migration task, the target format is one of Boomi REST integration patterns, FHIR R4 JSON, AWS Step Functions ASL JSON, or OpenAPI 3.0 with optional asynchronous extensions Although the surface formats differ, the underlying transformation problem is consistent: produce a modern, typed, well-structured artifact that stores the semantics of the legacy source.

4 Experiments

The framework was implemented within the Microsoft Azure ecosystem, using Azure AI Foundry as the core agent engine and Azure AI Search as the retrieval backbone. This choice was driven by enterprise-grade security, scalability, and integration with the organization's existing infrastructure. The language model powering all three baseline configurations is GPT-4.1, deployed via Azure OpenAI within the Foundry project; semantic embeddings were generated using the text-embedding-3-large model, deployed in the same project. All three baselines share this model, a sampling temperature of 0.0, and the same maximum response length.

The architecture of the experiment has three components: different knowledge sources, a retrieval engine, and a transformation agent. These components are shared to the extent each baseline uses them. B_0 uses none of these components. B_1 uses the knowledge base statically, embedded in the prompt at design time and not retrieved at runtime. B_2 uses all three layers.

4.1 Knowledge Base

The knowledge base consists of three different knowledge sources, and they are each hosted in Azure Blob Storage to be indexed by Azure AI Search using `text-embedding-3-large` vector embeddings. Having these knowledge bases allows the retrieval reasoning model to selectively query the most relevant source for each subquery instead of searching all sources from a single index.
The first knowledge source `ks1-legacy-schemas` stores legacy schema references: XSD, WSDL, XML, and JSON artifacts for the SOA Suite handbook corpus and the Capgemini dataset, together with FHIR R4 XML schema documentation for the healthcare leg. These resources provide the agent with direct access to source schema structure and concrete data examples.
The second knowledge source `ks2-target-profiles` stores structured natural-language descriptions of the modern target formats: Boomi REST connector specifications and JSON profile definitions, FHIR R4 JSON profile documentation, AWS Step Functions ASL conventions, and OpenAPI 3.0 patterns.
The third knowledge source `ks3-mapping-rules` stores field-level mapping rules, type conversion patterns, protocol mapping patterns (SOAP-to-REST, BPEL-to-ASL, async-SOAP-to-async-REST). The rules are written as structured markdown with explicit rule identifiers to enable both retrieval and citation in the agent's output.

4.2 Search Engine

Retrieval is performed through Foundry IQ, Microsoft's Agentic Retrieval engine, which operates as the sequential decision-making layer of the system. Rather than executing a single-pass retrieval over all knowledge sources, Foundry IQ uses GPT-4.1 as a reasoning model to autonomously decide which knowledge sources to query for a given input, formulate targeted subqueries, execute them in parallel, and merge the retrieved results before passing them to the transformation agent.

4.3 Transformation Agent

The transform agent was set under a structured Instructions block that specifies the agent's role, the workflow, and the output format. For each input migration task, the agent identifies the source fields and structures from the retrieved legacy schema knowledge, then finds the relevant target fields in the retrieved target profile. After that, it applies the mapping rules and transformation patterns to verify the result in the form of the target profile and to correct any inconsistencies before the final output.

4.4 Use Case Selection

The thesis evaluates five use cases, selected to probe distinct capabilities of the agent.

UC1 covers schema-level migration and is structured as **three complementary legs** that share the same evaluation framework but draw on different data sources.

- **UC1.1** is the academic benchmark drawn from the SOA Suite handbook ($n = 6$), with hand-prepared gold standards and a quantitative evaluator.

UC1.2 is the Capgemini production data leg ($n = 2$), validated through the company's integration tooling rather than through offline scoring.

UC1.3 is the FHIR R4 healthcare leg ($n = 10$), evaluating cross-domain transferability.

UC2 covers control-flow migration: translation of a synchronous BPEL process (QuickAndSlowProcess, cache-aside pattern) to AWS Step Functions ASL JSON. This use case probes whether the agent can translate control flow between two formal workflow languages with preservation of source-element provenance.

UC3 covers open-ended design: production of three alternative modern asynchronous REST design specifications (webhook, polling, server-sent events) from a legacy async SOAP service (AsynchKelvinToFahrenheitProcessor), with explicit rationale and a final recommendation. This use case probes whether the agent can reason about multiple legitimate alternatives.

The five use cases together span the quantitative-qualitative spectrum (UC1 and UC2 quantitative, UC3 qualitative) and the closed-open scope spectrum (UC1 single-artifact, UC2 single-process, UC3 open-ended design). Each use case is run on all three baselines (B0, B1, B2) where data permits; UC1.2 was run on B2 only due to the small sample size and the focus on industrial validation rather than baseline comparison.

4.5 Evaluation Methodology

Each use case uses a dedicated evaluator chosen to match its task characteristics. The metrics differ across use cases because what counts as a correct migration differs across them; what is comparable is the spread of scores between B0, B1, and B2 within each use case, not the absolute scores between use cases.

For UC1.1 and UC1.2, the evaluator measures schema validity, field coverage, type accuracy, and operation mapping at the test-case level, and then aggregates over the test set.

For UC1.3, the evaluator measures schema conformance, field key accuracy, value accuracy, and type coercion accuracy, the last being the FHIR-specific differentiator: whether boolean and numeric values are correctly typed as JSON boolean and numbers rather than strings.

For UC2, the evaluator measures ASL validity, state extraction precision and recall against the gold BPEL activity roles, control-flow reachability, task binding fidelity, and synchronous-response semantics. Each evaluator was self-tested on the gold standard against itself before being used; the self-test score is reported in the corresponding results subsection.

UC3 admits no single canonical answer, because the migration of an asynchronous SOAP service to modern REST can be defended in multiple patterns. A six-dimensional rubric was used: coverage of three designs, OpenAPI 3.0 specification validity, trade-off articulation, risk identification, source-contract fidelity, and the recommendation quality. Each dimension was scored on a 0/1/2 ordinal scale per system, with a maximum of 12. Each output was read in full and scored across the six dimensions.

For borderline decisions regarding more subjective dimensions (trade-off specification and recommendation quality), the rubric for the specific dimensions was reviewed, and the most conservative decision was made based on the given output.

The output of Capgemini data (UC1.2) uses domain-specific validation instead of academic test cases. After the agent produced JSON output for each of Capgemini's legacy data inputs, the

output was imported to the Boomi platform. Boomi is an integration platform used by the company and a downstream tool for mapping configuration and many more integration tasks. In this case, as discussed with the company, the criterion of success was that every field could be mapped in the target Boomi process. This case tests not only the structural correctness of the outputs but also the practical usability of the outputs in the operational environment.

5 Results

This section reports the evaluation of the agent across the five use cases defined in Section 4.4. Each use case is run on the three baselines B_0 (LLM-only), B_1 (Traditional RAG), and B_2 (the agent) where the data permits, and the results are presented in the order in which the use cases were introduced.

5.1 UC1 — Schema-Level Migration

UC1 evaluates the agent's ability to perform schema mapping. This is the main task for which the knowledge base was created. The evaluation is divided into three parts: an academic benchmark (UC1.1), industrial validation on Capgemini data (UC1.2), and cross-domain transferability on healthcare data (UC1.3).

5.1.1 SOA → Boomi REST Academic Benchmark

The primary quantitative benchmark uses six test cases drawn from the Oracle SOA Suite 12c Handbook companion source repository [2]. Each test case has an XSD schema, a WSDL service contract, an example XML instance, and a hand-prepared gold-standard OpenAPI 3.0 JSON representing the equivalent modern Boomi REST artifact.

Table 1: UC1.1 evaluator output. Averages over $n = 6$ test cases.

Metric	B_0	B_1	B_2
Schema validity	100.0%	100.0%	100.0%
Field coverage	80.4%	80.4%	**83.7%**
Type accuracy	59.3%	55.1%	**61.7%**
Operation mapping	50.0%	**83.3%**	50.0%
Overall	72.4%	**79.7%**	73.9%

The evaluator computes four metrics per test case and aggregates over the six cases. Schema validity verifies that the predicted JSON parses as OpenAPI 3.0. Field coverage measures the fraction of gold-standard fields that appear in the prediction. Type accuracy measures whether matched fields have the correct JSON type. Operation mapping measures whether the HTTP verb assignment and the path match for each operation, scored as the mean of method match and path match. The aggregate score combines the four metrics with equal weighting. Table 1 reports the per-system results.

[2] https://github.com/lucasjellema/soasuitehandbook

B_1 leads at 79.7%, while B_2 (73.9%) and B_0 (72.4%) trail by roughly six to seven percentage points and sit within 1.5 points of each other. The overall ranking is driven by a single metric: B_2 leads on both field coverage and type accuracy, but B_1's decisive advantage on operation mapping outweighs both.

B_2 achieves the highest field coverage (83.7% against 80.4% for both other systems) and the highest type accuracy (61.7%). Notably, static retrieval alone does not produce this gain: B_1 matches B_0 exactly on field coverage. Only the agentic configuration recovers the additional XSD-level field and type information.

The metric that produces B_1's overall lead is operation mapping, where B_1 reaches 83.3% while B_0 and B_2 both sit at 50.0%. The decomposition is informative: the operation-mapping score is the mean of method match and path match, and the 50.0% figure for B_0 and B_2 corresponds to consistently correct HTTP verb assignment (100% method match) paired with a path that does not match the gold (0% path match). The failure mode for B_0 and B_2 is therefore path style rather than verb selection: the systems produce REST paths that map operations to logically reasonable but syntactically different URLs than the gold standard prescribes (for example, `/orders/{orderId}/items` versus `/orderItems`). B_1 scores higher on this metric because the static rules embedded in its prompt include the Boomi path conventions explicitly, so its paths align with more often.

The overall spread of roughly seven percentage points reflects the nature of the task. Schema-level field mapping is a task GPT-4.1 already does well from parametric knowledge; the retrieval-augmented systems gain over the baseline only to the extent that they apply the handbook's domain-specific conventions (Boomi path styles, verb prefixes, fault mapping patterns), which the model would otherwise have to guess. The agent-augmented configuration (B_2) does not reach the static-retrieval configuration (B_1) overall on this task: B_2's gains in field coverage and type accuracy are outweighed by B_1's advantage in path conventions, leaving B_2 only 1.5 points above the LLM-only baseline.

5.1.2 Capgemini Production Data — Industrial Validation

UC1.2 is the original motivating experiment of the thesis. Two production legacy XML files were provided by the Capgemini integration cluster, each accompanied by its XSD schema and WSDL service contract, and the agent was tasked with producing the equivalent modern JSON for each. Because $n = 2$ does not support a comparison across the three baselines, only B_2 was evaluated; the purpose here is industrial validation rather than baseline ranking.

The validation follows operational Capgemini practice rather than offline field comparison. Each agent output was imported into Boomi, the iPaaS tool used by Capgemini, and configured into a downstream mapping through its visual interface. An output is considered correct when every field maps to a destination field, type compatibility holds throughout, and the resulting process executes without configuration errors; this tests not only structural correctness but the practical usability of the output in the environment for which it is intended. Both files passed: no type mismatches occurred, no missing fields were reported by the Boomi validator, and manual inspection confirmed

that all source fields are represented, that the Dutch comma decimal format is correctly converted to the international point format, and that the result follows the conventions used elsewhere in the Capgemini Boomi environment.

UC1.2 therefore yields a quantitative result of 100% accuracy on $n = 2$ alongside the qualitative result of a successful Boomi import. Boomi mapping validation is the check a Capgemini integration engineer applies before approving any conversion for production, so passing it on two real artifacts is meaningful evidence of operationally usable output. To validate the accuracy of this mapping, the Boomi process was deployed behind a Web Listener and exposed as a REST service. The legacy XML was sent to it as the body of an HTTP POST request from an external client. It also authenticated with Basic Auth against a Capgemini test account. The endpoint returned the converted JSON directly, and the response matched the design-time output: every field was present, the numeric values carried the international decimal format, and the document was well-formed. This confirms that the mapping behaves correctly not only inside the Boomi designer but also when invoked as a live service, which is the form in which it would be consumed in production.

5.1.3 FHIR R4 XML → JSON — Cross-Domain Validation

The third leg of UC1 extends the schema-migration evaluation into a new domain. UC1.1 demonstrated the agent's behaviour on legacy Oracle SOA to Boomi REST mapping, the corpus on which the knowledge base was built. UC1.2 demonstrated industrial validity on Capgemini-provided production data. UC1.3 tests cross-domain transferability: whether the same agent, with the same Instructions block and the same retrieval loop, works on a schema migration of another setting, healthcare interoperability, rather than enterprise integration.

The use case is FHIR R4 XML to JSON conversion. FHIR (Fast Healthcare Interoperability Resources) is the dominant standard for healthcare data exchange. The R4 specification publishes each resource definition in two equivalent serialisations, XML and JSON, and many existing healthcare systems hold their FHIR records as XML while modern APIs have largely converged on JSON. The migration task is direct: parse the XML, apply FHIR's R4 XML-to-JSON mapping rules, and emit valid JSON.

The XML sources are drawn from the official HL7 FHIR R4 specification examples corpus. Each test case has a gold-standard JSON, structurally validated against the FHIR R4 JSON schema and checked manually for correct type coercion (XML's `value="true"` becoming JSON's `true`, `value="1"` becoming `1`, repeated elements becoming JSON arrays). Each test case is rich enough to be informative — the Patient case alone contains roughly fifty fields across nested structures, including two FHIR primitive extensions and a deeply nested contact subresource.
Table 2 reports the per-system results.

Table 2: UC1.3 evaluator output. Averages over $n = 10$ test cases.

Metric	B_0	B_1	B_2
Schema conformance	100.0%	100.0%	100.0%
Field key accuracy	100.0%	100.0%	99.32%
Value accuracy	100.0%	96.19%	99.32%
Type coercion accuracy	100.0%	100.0%	100.0%

Four metrics are computed per test case and then averaged. Schema conformance checks whether the predicted JSON's `resourceType` matches the gold's and whether the document parses as valid JSON. Field key accuracy measures, of the dot-notation field paths in the flattened gold (e.g. `identifier[0].type.coding[0].code`), what fraction appear in the prediction. Value accuracy measures of those matched paths, what fraction have matching values. Type coercion accuracy measures, among gold fields whose value is a boolean or a number, what fraction the prediction also typed correctly, rather than as a JSON string.

The fourth metric is the FHIR-specific differentiator and the most informative on this task. In FHIR XML, every primitive value is serialised as an attribute string (`value="true"`, `value="1"`). A correct JSON conversion must coerce these into real booleans and real numbers; strings will validate as JSON but will fail downstream healthcare integrations that expect typed values. A self-test of the evaluator on the gold standards against themselves yields 100% on all four metrics.

The ten source XML files are drawn from the official HL7 FHIR R4 specification examples. These files have been publicly available since the FHIR R4 ballot in 2018. They are among the most widely referenced examples in healthcare interoperability. There is a meaningful probability that GPT-4.1's training corpus includes either these XML files, their JSON counterparts, or both.
If the training corpus contains the canonical JSON form of the Patient example, then a system asked to convert that XML to JSON can succeed by recall from parametric memory rather than by applying transformation rules. The XML input cues the model to reproduce the corresponding JSON output, and the transformation never has to happen.

There is a small piece of indirect evidence. B_0 scored 100% on value accuracy — an exact-string match between every flattened field in every prediction and every gold. B_1 scored 96.19%. The slight drop in B_1 is explainable because when the prompt embeds explicit transformation rules and asks the model to apply them step by step, the model is nudged away from pattern-completing from memory and toward rule application. Rule application introduces minor formatting differences (a trailing zero on a decimal, a whitespace choice, a coercion that is technically correct but does not exact-string-match) that exact comparison penalises. This is the pattern one would expect if B_0 is largely reproducing memorised output while B_1 is doing more transformation work.
UC1.3 is therefore interpreted as a sanity check on cross-domain applicability. The agent does not break any artifacts of FHIR, and it produces well-formed FHIR R4 JSON outputs. However, this case cannot be a discriminating benchmark of the retrieval and agent layers. Therefore, the thesis comes with UC2 (BPEL to ASL, where the spread is wide and provenance discipline explains it) and UC3 (async pattern design, where the spread is narrow and source-fidelity discipline differentiates qualitatively).

One result worth reporting on its own is the 100% type coercion accuracy across all three systems. Type coercion is the specific challenge that distinguishes correct FHIR conversion from naive XML-to-JSON conversion: a converter such as `xmltodict` emits every value as a string (`"active": "true"` rather than `"active": true`), which is enough to break FHIR-aware consumers downstream. All three systems instead produce the correct typed values across every boolean and numeric field in the test set. For B_0 likely through training exposure combined with its ability to emit typed JSON, and for B_1 and B_2, because the retrieved rules and the knowledge base both call out the requirement explicitly. The practical implication: a naive `xmltodict.parse` would fail this metric across the board, so even without ranking the three systems, the use case shows that LLM-based conversion outperforms naive deterministic conversion on the FHIR-specific aspects of the task.

Four threats are flagged. The training data exposure discussed above is the single largest one, and UC1.3 should not be read as evidence of retrieval value for novel FHIR data. The test set is informative but small at $n = 10$; a larger evaluation with adversarially modified artifacts — renamed fields, novel resource combinations, and out-of-distribution extensions — would be needed for a quantitative cross-domain claim and is left to the concluding chapter. The gold standards are hand-prepared and reflect one reading of the FHIR R4 JSON specification, particularly for edge cases such as primitive extension placement (FHIR's `_fieldName` properties). Finally, the value-accuracy metric uses trimmed string comparison, so a coercion-correct `"1.0"` against a gold `1` need not match — the most plausible explanation for B_1's 96.19% value accuracy, which a more type-aware comparator would likely narrow.

5.2 UC2 — BPEL Orchestration to AWS Step Functions

UC1 measured how well the agent maps individual fields between schemas that the agent has visibility into on both sides. UC2 tests something different. It tests whether the agent can translate *control flow* between two formal workflow languages that share concepts but differ in syntax and execution semantics. Our source language is BPEL, which is the orchestration language Oracle SOA Suite uses to describe the steps of a business process. The target is Amazon States Language (ASL), the JSON format that AWS Step Functions also uses to describe step-by-step workflows in the cloud. Both languages express a sequence of steps for the workflow with branches, waits, and external calls. However, BPEL processes use *activities*, ASL workflows use *states* for the same workflow operations. Additionally, some BPEL constructs such as scopes, compensation handlers, and link types have no direct ASL equivalent. This necessitates actions such as re-expression or a drop in the transformation. The task is an example of a legacy activity-to-cloud state transition but not a field-to-field mapping.

The source process is `QuickAndSlowProcess.bpel`, this time taken from chapter 9 of the Oracle SOA Suite handbook corpus[3]. It implements a *cache-aside* pattern, which is a common lookup strategy. When a request arrives, the process must check the cache for an answer. If the answer is already there, it should immediately be returned. If it is not, the process does the slow work and stores the result in the cache to return it. A five-second `<wait>` stands in for the expensive

[3] https://github.com/lucasjellema/soasuitehandbook

computation.
The full activity tree contains seven distinct named activities:

- `SetCacheTag` — an assign;
- `RetrieveFromCache` — a `bpelx:call` to the CoherenceCacheRetriever;
- `IfDataInCache` — a conditional with two branches;
- `SetIntermediateResult` — an assign that writes a literal result;
- `ArtificialLongProcessing` — the five-second wait;
- `PutPayloadOnCache` — a `bpelx:call` to the CoherenceCacheWriter;
- `AssembleResponse` — a final assign that builds the response string via XPath `concat`.

The process begins with a `<receive>` and ends with a `<reply>` on the same partner link. So that the caller waits for the answer within a single request-response exchange without another callback.

This BPEL was selected because the cache-aside structure already has five of the ASL state types in a single small process (Pass, Task, Choice, Wait, terminal), making it reliable to prepare a gold standard from.

To understand clearly which state in the new ASL matches the original BPEL, an evaluation is necessary. The evaluator was therefore designed to reward two things at once: that the ASL is correct and that each state can be traced back to its source activity. This evaluator should be able to grade an arbitrary ASL JSON against the gold design without using the same gold standard's state name. Any sensible name should be acceptable, but that gives credit to the *provenance* of the migrated artifact.

Five weighted metrics are computed:

- **ASL validity** (10%) — the JSON parses, `StartAt` and `States` are present, and at least one state terminates the machine.
- **State extraction F1** (30%) — for each of the seven gold roles, the evaluator looks for a predicted state of the matching ASL Type whose name equals the source BPEL activity name verbatim, or whose `Comment` field cites it; a state may match at most one role, with precision over the predicted state set and recall over the gold role set.
- **Control flow** (30%) — for each of the seven gold transitions between roles, whether a path can be found in the predicted state graph from the source role's mapped state to the target role's mapped state, allowing extra intermediate states and bounding reachability to six hops.
- **Task binding** (20%) — whether the two cache-related Task states carry `Resource` and `Retry` blocks.
- **Synchronous semantics** (10%) — whether the role-mapped response-assembly state produces a string via `States.Format` or an equivalent construct.
  Table 3 shows the per-system results.

Table 3: UC2 evaluator output.

Metric	B_0	B_1	B_2
ASL validity	100.0%	100.0%	100.0%
State extraction Precision	0.0%	70.0%	77.8%
State extraction Recall	0.0%	100.0%	100.0%
State extraction F1	0.0%	82.4%	87.5%
Control flow (edges)	0.0%	100.0%	100.0%
Task binding fidelity	0.0%	100.0%	100.0%
Synchronous semantics	0.0%	100.0%	100.0%
Overall	**10.0%**	**94.7%**	**96.2%**

The headline is the gap: B_0 scores 10% while B_1 and B_2 score 94.7% and 96.2%. This spread of roughly 85 points is by far the largest

B_0 has correct ASL with parsing, and it has the expected nine states. However, the control flow internally consists of correct external call states and final states; it did not give the accurate labelling. B_0 renamed every state in its own words (`SetCacheTag` became `ComputeCacheTag`, `RetrieveFromCache` became `CacheGet`, `IfDataInCache` became `CacheHitChoice`) and wrote comments that describe what each state does rather than which BPEL activity it came from. The evaluator cannot match any of these states to a source activity, and because every other check (control flow, task binding, semantics) is measured over the matched states, they all collapse to zero as well.

To confirm that the underlying ASL is in fact correct, the original B_0 output was taken and only relabelled: each state was renamed to its gold counterpart and its comment rewritten to cite the source activity, with no change to any state type, ARN, Retry block, parameter, or transition. After relabelling, the score rose to 92%. Eighty-two of the missing ninety points are therefore not about whether the translation is correct but about whether it is *traceable*: B_0 produced a correct artifact with no provenance metadata, whereas B_1 and B_2 keep the BPEL activity names verbatim and add `Comment` fields such as `"BPEL <assign name='SetCacheTag'>"`. The evaluator may be too strict, but in this setting it is important that the output works and can be traced back to the source. Otherwise it requires a manual annotation pass before it can be deployed.

B_0 (LLM only) generated nine states characterized by standard internal capitalization naming conventions and operational, rather than provenance-oriented, `Comment` fields. Structurally, B_0 partitioned the response assembly phase into two distinct states—`AssembleResponseFromCache` and `AssembleResponseFromLocal`—instead of converging both cache branches into a single, unified state. Additionally, it concludes with a `Succeed` state instead of specifying `End: true` on the final Pass state. Ultimately, both of these structural modifications constitute stylistic choices that are valid in ASL and guarantee identical runtime behaviour.

B_1 (Traditional RAG) produced states with names accurately matching the BPEL activities and comments citing them in prose. One extra pass-through state (`ReturnResponse`) with no gold counterpart is the reason precision is 70% rather than 100%. Task `Resource` ARNs use the literal Lambda

pattern (`arn:aws:lambda:...`) rather than a service integration form (`arn:aws:states:::lambda:invoke`); both are acceptable.

B_2 (Schema-Mapping Agent) produced nine states with exact BPEL names and a more compact `Comment` citation that also names the BPEL element type, e.g. `"BPEL <extensionActivity><bpelx:call> RetrieveFromCache"`. B_2 produces one fewer spurious state than B_1: it does not add the pass-through state (`ReturnResponse`) that B_1 produces. This is the main reason for the precision difference between the two baselines (77.8% versus 70%).

Three key points are flagged regarding these results:

- **Strict Evaluation Criteria:** A lenient evaluator comparing only ASL Type sequences would have scored B_0 much higher. Consequently, the headline numbers should be read in conjunction with the relabelled-B_0 result (92%). This strict approach is deliberate, since traceability represents a genuine success criterion in industrial migration practice.
- **Sample Size Limitations:** Because UC2 evaluates a single BPEL process ($n = 1$), the identified pattern can only be claimed for a representative cache-aside synchronous process from the same corpus as UC1, not for BPEL workflows in general.
- **Gold Standard Subjectivity:** The gold standard inherently encodes specific opinions about idiomatic ASL, such as configuring `TimeoutSeconds: 30`, adopting the service-integration ARN form, and defining the `UseCachedResult` intermediate state. Another author could compose an equally valid gold standard against which the same agent outputs would yield different scores.

The UC2 result can be interpreted in two ways, depending on the baseline in focus. Against the LLM-only baseline, it is a dramatic demonstration of retrieval value: a gap of roughly 85 points, the largest any use case produces, showing that discipline around migration provenance is something the plain LLM does not bring on its own.

Against the Traditional RAG baseline, it is incremental: B_2 scores 1.5 percentage points above B_1, an edge that comes entirely from B_2 emitting one fewer spurious state than B_1 and so scoring slightly higher on state-extraction precision. The agent's contribution is therefore large at the methodological level (relative to no retrieval) and small at the architectural level (relative to static retrieval). It is also worth noting that UC2 is where the knowledge base is most visible, yet teaches the least surprising lesson: the rule that every ASL state should name its source BPEL activity is the kind of guidance a migration handbook prints on its second page. B_0 missed it because nothing pointed it out; B_1 and B_2 followed it because they were pointed at it.

5.3 UC3 — Asynchronous Pattern Design

UC1 and UC2 evaluated the agent on tasks with at least one defensible reference answer, and each of them was scorable against a gold standard. Asynchronous SOAP services do not admit a single canonical modern equivalent. The same legacy contract can be migrated in at least three ways: a webhook design, in which the server delivers the result by calling back an address the client supplies; a polling design, in which the client repeatedly asks the server whether the result is ready;

or a server-sent-events (SSE) design, in which the server keeps a connection open and streams the result to the client. Each carries different operational tradeoffs, and each is defensible under appropriate conditions. UC3 therefore tests whether the agent can reason about multiple legitimate alternatives, articulate their tradeoffs, and justify a recommendation grounded in the source contract.

The source artifact `AsynchKelvinToFahrenheitProcessor.wsdl` is from chapter 23 of the Oracle SOA Suite handbook corpus[4]. It defines an asynchronous one-way SOAP service: a provider port type with a single input-only operation (`requestKelvinToFahrenheitConversion`) paired with a requester-side callback port type (`replyKelvinToFahrenheitConversion`) bound by a BPEL `partnerLinkType` with two roles.

The three systems used the same model (GPT-4.1), temperature (0.0), and response limit as UC1: B_0 as a plain LLM completion, B_1 with the relevant async-SOAP-to-REST rules embedded as literal prompt text, and B_2 inside the same Azure AI Foundry agent as UC1 (identical instructions, same knowledge base, and no per-use-case tuning).
All three were asked for three OpenAPI 3.0 specifications, each with a rationale and risks, plus a recommendation. Requiring three designs rather than one is deliberate: it removes the single-answer assumption and surfaces the agent's ability to reason about alternatives. The six dimensions were scored on an ordinal 0/1/2 scale.

- **D1 coverage:** Three genuinely distinct patterns rather than variants of one.
- **D2 spec validity:** Each OpenAPI document well-formed with its pattern-specific structure, e.g. the SSE design's `text/event-stream` content type.
- **D3 tradeoff articulation:** Each rationale cites at least one operational tradeoff such as cost, network topology, idempotency, or scaling.
- **D4 risk identification:** Pattern-specific risks.
- **D5 source fidelity:** Faithful preservation of the input/output semantics, the asynchronous nature, and the two-port-type structure.
- **D6 recommendation quality:** A final recommendation grounded in the source contract and explicit about why the rejected designs were rejected.

Table 4 presents the scoring results.
The headline result is convergence rather than separation: all three systems recommended the polling pattern, and the spread between B_0 (10/12) and B_1/B_2 (both 11/12) was a single rubric point. Unlike UC1.1, where metric scores separated the systems (B_0 72.4%, B_1 79.7%, B_2 73.9%), and unlike UC2's large evaluator-driven gap, UC3 shows the three systems arriving at essentially the same answer.

The scoring table conveys the magnitudes; the following observations convey the differences in kind, drawn from reading the three outputs side by side.

[4] https://github.com/lucasjellema/soasuitehandbook

Table 4: UC3 per-dimension scores. Each cell is the system's score on the corresponding dimension; the final row gives totals.

Dimension	B_0	B_1	B_2
D1 Coverage of three designs	2	2	2
D2 Spec validity	1	1	1
D3 Tradeoff articulation	2	2	2
D4 Risk identification	2	2	2
D5 Source fidelity	1	2	2
D6 Recommendation quality	2	2	2
Total (max 12)	**10**	**11**	**11**

Source-contract fidelity serves as a key differentiator between the systems, even when their quantitative scores are close:

- **Baseline B_0 Deviations:** Although it produced a usable webhook design, B_0 failed to align with the source contract. First, it introduced a meaningless `/callback-example` path, ignoring that the real callback URL is client-supplied at request time. Second, its recommendation referenced "the BPEL stateful process model," despite the source artifact containing no BPEL, only a WSDL async port type pair.
- **Baseline B_1 Alignment:** B_1 avoided these errors by enumerating the two source operations in its validation report and directly tying them to their respective REST counterparts.
- **Baseline B_2 Structural Reasoning:** B_2 also avoided B_0's mistakes, noting that polling closely matches the legacy contract. It uniquely mapped the one-way-plus-callback shape to a POST-plus-GET structure, making it the only baseline to reason about the source by its structural shape rather than just operation names.

This behaviour demonstrates a quality dimension that automated metrics cannot capture: it measures whether the source artifact was genuinely referenced during the generation process, rather than just evaluating if the output is abstractly defensible.

The output of the Agentic RAG baseline concluded with an explicit statement tracing its mapping rules and schema details back to the retrieved ks3 and protocol pattern rules. We can analyse from here that the retrieval was active during the generation. This attribution was made at the broad knowledge-container level instead of citing individual rule identifiers. From the output alone, it is impossible to determine whether this reflects an inherent choice by the agent to cite file granularity rather than rule granularity.
For this use case, it is clear that B_2's retrieval was demonstrably active during the run, but the granularity of evidence available to an external auditor is coarse.
The qualitative methodology applied here introduces specific threats to validity that were not present in the quantitative evaluation of UC1. First, this use case evaluates only a single asynchronous SOAP service ($n = 1$). Consequently, it should be interpreted as a focused case study of a single example rather than a generalized claim about asynchronous-to-REST migrations. This single-source scope was deliberately chosen to keep the qualitative evaluation dimensions manageable.

The ordinal 0/1/2 scale introduces potential bias, especially when evaluating complex dimensions, such as D3 and D6.

B_2 was evaluated using the knowledge base in the original UC1 state. Although a dedicated, asynchronous-specific corpus was prepared, its formal re-evaluation is left for future work. Notably, this constraint makes the convergence finding more compelling, as B_2 successfully matched B_1 (which held the relevant rules directly in its prompt) using only a general corpus.
UC1 showed that the agent matches Traditional RAG on a precise mapping task, with both ahead of the LLM-only baseline. UC3 shows the same pattern in an open-ended design task with no single correct answer. The agent's recommendation is as good as the Traditional RAG recommendation and slightly better than the LLM-only baseline. The difference appears in two qualitative aspects that UC1's metrics cannot measure: whether the source contract is followed faithfully and whether the recommendation is explicitly grounded in it.
Together, the two use cases mark the boundaries of what the agent can and can not do. UC1 shows that it can be precise when there is a clear definition of the correct output. UC3 showed it can produce an adequate answer when the deciding criteria are more disciplined, rather than its correctness.

6 Discussion

The five use cases evaluated whether retrieval-augmented agentic reasoning helps with legacy-to-modern migration from different angles. With different datasets, different tasks, and different answers, we can find when retrieval and agentic loops add value and when they do not.
Figure 1 summarises the results spread across the four out of five use cases. UC1.2 is validated operationally and is not shown.

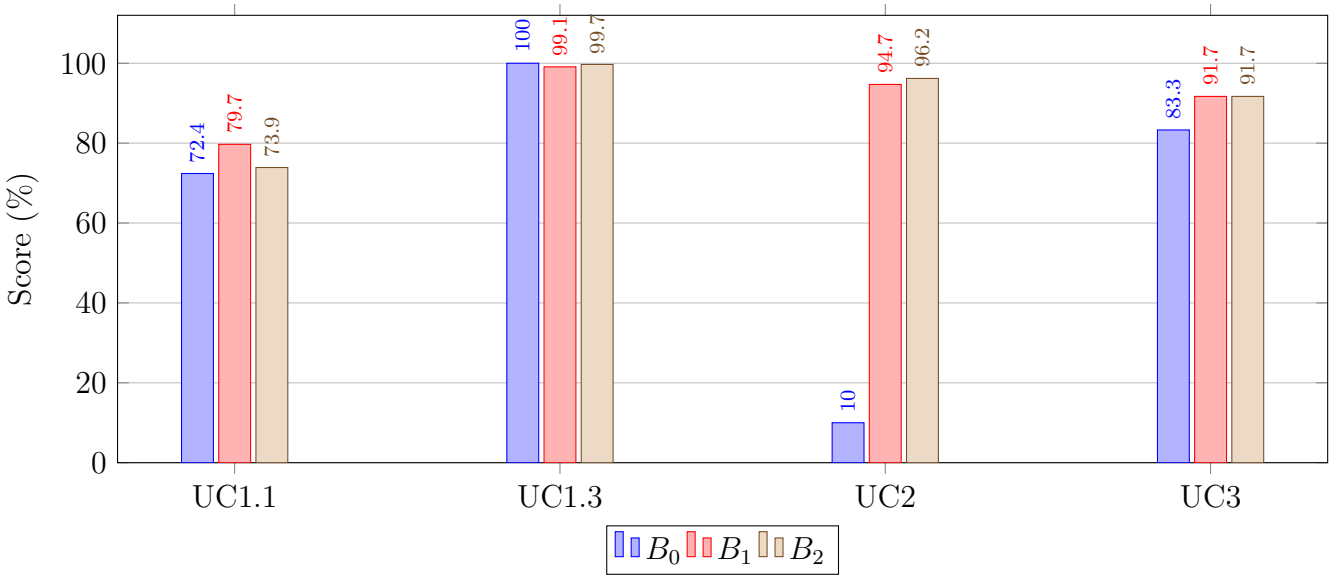

Figure 1: Per-use-case results summary

UC1.1 uses the four-metric aggregate (schema validity, field coverage, type accuracy, operation

mapping)
UC1.2 uses enterprise evaluation (not shown in this figure)
UC1.3 uses field-path matching with type coercion
UC2 uses provenance-aware ASL state extraction
UC3 uses a qualitative rubric with six dimensions.

Combining the results, a heuristic emerges for predicting when retrieval-augmented systems will outperform an LLM-only baseline.

Retrieval value is high when the task requires discipline external to the content. UC2 is the clearest example. The semantic translation between BPEL and ASL is well within GPT-4.1's performance. B_0's output, with its names changed to the correct names, scored 92% on the same evaluator. What B_0 lacked was the discipline to preserve source activity names and write traceable `Comment` annotations. The retrieval-augmented agents had access to a rule prescribing exactly this, and they followed the rules. The rule was describing specific migration practices, and it does not emerge from the model's pre-training alone.

Retrieval value is moderate when the task uses specialised vocabulary that the model knows partially. UC1.1 fits here. SOA WSDL elements, REST OpenAPI structures, and the mapping between them are well represented in pre-training, but the specific conventions the migration handbook uses (verb prefixes, path style, fault mapping) are not. B_1 outperforms B_0 by a clear margin (7.3 points) and B_2 by a small one (1.5 points), precisely to the extent that they apply the handbook's conventions.

The retrieval value is low when the task is within the parametric capability. UC3 fits here. Designing three alternative async REST patterns and writing rationales is the kind of explanatory reasoning that GPT-4.1 does well, even without retrieval. B_0 did not fail at producing three patterns, writing tradeoff analyses, or recommending one of them. This is why, in AI Agent architectures, LLMs serve as the primary decision-making engine [SRK25]. It only failed at source-contract fidelity, and a more specific deficiency that did not translate into a large rubric-score gap.

Retrieval value cannot be measured when training data leakage is plausible. UC1.3 fits here. The HL7 FHIR R4 spec examples are publicly available and likely present in GPT-4.1's training corpus. A system that succeeds because it remembers the gold standard from training does not need retrieval to succeed.

These four conditions aren't mutually exclusive. UC1.1 mostly follows the specialised-vocabulary pattern, but it also shares a bit of the discipline pattern through its HTTP verb conventions and fault mapping. Similarly, UC3 fits the within-parametric capability pattern, though the source-fidelity results show a subtle layer of discipline during the qualitative analysis. Finally, these categories work better as a practical guide rather than a strict, clean partition.

Another condition was UC1.2 (Capgemini-provided production data, B_2 only, $n = 2$, manual validation at 100% accuracy). This condition does not fit the spread analysis because it ran only in an integration tool. For this study, it adds external validity alongside the academic value of the other results. The used dataset was not publicly available, and 100% manual accuracy on those two

production schemas is the strongest available evidence that the agent's behaviour is not specific to the training-data-overlap conditions of the other use cases.

The other question in this experimental setup was about the value that Agentic RAG adds to legacy-to-modern migration scenarios. The measured output differences are not so high, but the difference is in how the setup behaved beyond a list of tasks. Each B_1 prompt in this thesis was built knowing in advance which rules the use case would need. This is not suitable for production-level because classifying every new job by hand is fragile, and putting the entire rule corpus into every prompt eventually overruns the context window. The agent picks the right rules itself at runtime, which allows a single configuration to handle tasks that have not been pre-labelled.

In some migration scenarios, particularly in tasks where the target structure must be verifiable against the source (like UC2), the agent's advantage lies in scalability rather than accuracy per task. LLMs are able to do well on open-ended designs, canonical-format conversions, and common API patterns if the risk of over-engineering is not taken into account.

7 Conclusions

In this thesis, we attempted to discover how retrieval-augmented agentic reasoning would improve legacy-to-modern migration. Therefore, the main research question is as follows:

How can intelligent agents be designed to autonomously interpret and transform heterogeneous data formats during legacy-to-modern system migration, leveraging Retrieval-Augmented Generation to map disparate schemas and protocols with minimal human intervention?

The answer developed and evaluated in this thesis is a domain-specific agent that combines a retrievable knowledge base of migration rules, an iterative agentic workflow, and a general-purpose language model as a reasoning engine. This agent was compared with an LLM-only baseline (B_0) and a Traditional RAG baseline (B_1). The testing environment was across five use cases spanning schema mapping, industrial validation, orchestration translation, and open-ended design.

The Agentic RAG design outperforms when the tasks require knowledge or discipline that the model does not bring itself; such as the domain conventions of the schema benchmark in UC1.1, and most decisively the migration provenance requirements of the orchestration task in UC2, where preserving source activity names and traceable annotations produced the largest separation observed in the study.

The agentic workflow enables this through the agent's sequential design:
`Plan` → `Retrieve` → `Transform` → `Verify` → `Correct`
The instruction set directs the agent to apply retrieved rules consistently rather than mentioning them once and dropping them; the extent to which the verification steps specifically drive this behaviour could not be isolated (Section 6). Where the task lies within the model's parametric capability (UC1.3, UC3), this loop adds little that the metrics can detect, although the qualitative findings on UC3 show that B_2 still exercises more source-fidelity discipline than the LLM-only

baseline.

Additionally, industrial validity was established by the Boomi platform in Capgemini settings. We can conclude that the transformation can be done with minimal human intervention; however, validations still remain human and tool-driven.

The first sub-question asked which stages of legacy-to-modern migration intelligent agents can support, and in which roles.

The evaluation maps these capabilities onto three distinct stages:

- **Schema-level migration (5.1):** The agent acts as a transformer, converting legacy XML/XSD structures into modern typed JSON.
- **Orchestration migration (5.2):** It acts as a translator and annotator, converting control flow between two workflow languages while keeping each target state traceable to its source activity.
- **Design-stage migration (5.3):** It acts as an advisor, producing alternative modern designs with their tradeoffs and a recommendation grounded in the source contract.

The second sub-question asked how iterative retrieval, reasoning, and transformation mechanisms contribute to accurate mapping. Two mechanisms proved decisive across the experiments:

- **Runtime retrieval as routing:** The agent fetches the rules a given task needs without those rules being known in advance. This is what allows a single agent configuration to handle different task types at accuracy comparable to per-task Traditional RAG prompts, without manual pre-description.
- **Rule-following as discipline:** The largest measured gains came not from better translation but from the agent applying retrieved rules that the model would not otherwise follow, particularly provenance annotation. UC2 demonstrates this most clearly: B_0's ASL output was structurally correct but failed at traceability, while B_1 and B_2 followed the same retrieved rule and produced auditable output.

The iterative verification and correction steps contributed less than anticipated at the measured level: on no use case did the agentic loop produce output noticeably better than static retrieval on the metrics. This is treated as a boundary of the evidence rather than as proof that the loop is unnecessary, for the reasons set out in Section 6.

Taken together, the answer to the main question is that an effective migration agent is designed around an explicit, retrievable rule corpus rather than around model capability alone. The model supplies the translation, the knowledge base supplies what correct migration means in the target organisation, and the agent's contribution is to bring the two together at runtime, task by task, without per-task engineering.

This division of responsibility also indicates when each design tier is appropriate. A plain LLM is sufficient when the task carries no specific requirements. A static retrieval design suffices when the

task follows a fixed set of conventions known in advance. The agent design earns its operational cost on tasks that must be traceable, auditable, and conformant to rules the model cannot guess on its own. The conditions of this study showed retrieval-augmented agentic reasoning to be most decisive.

8 Limitations and Future Work

8.1 Limitations

Several limitations constrain the interpretation of the results presented in this thesis.

1. **Sample size.** With the exception of UC1.1 ($n = 6$) and UC1.3 ($n = 10$), each use case evaluates a small number of artifacts (UC2, $n = 1$; UC3, $n = 1$; the Capgemini industrial validation, $n = 2$). These sample sizes support pattern observation but not population-level statistical claims.

2. **Circularity between gold standards and the knowledge base.** The gold standards for UC1.1 and UC2 were derived from the same source documents used to construct the knowledge base, which limits the extent to which retrieval performance can be distinguished from alignment between the gold standard and the retrievable content.

3. **Training-data exposure.** The FHIR R4 examples used in UC1.3 have been publicly available since 2018 and are likely present in GPT-4.1's training corpus, confounding the evaluation of retrieval value on this use case.

4. **No instrumentation of the agentic workflow.** The VERIFY and CORRECT stages were not separately instrumented, and their individual contribution could not be isolated from that of retrieval.

5. **Single model and retrieval engine.** All experiments use GPT-4.1 and Microsoft's Foundry IQ retrieval engine; results may not generalize to other language models or retrieval architectures.

8.2 Future Work

The limitations above motivate the following directions.

1. **Larger industrial sample** : A larger volume of Capgemini-provided production schemas would allow the industrial validation to be evaluated with the same rigor as the academic benchmarks, bridging the gap between the academic benchmarks and real production migration.

2. **Scaling UC2 and UC3** A broader dataset of BPEL processes of varying complexity and several asynchronous services is needed to test the provenance and source-fidelity findings as patterns rather than isolated case studies.

3. **Adversarial cross-domain evaluation** Modified FHIR artifacts — renamed fields, novel resource combinations, out-of-distribution extensions — would measure retrieval value on genuinely novel healthcare data.

4. **Ablation of the agentic workflow** (Limitation 4). Instrumenting the VERIFY and CORRECT stages individually, and comparing configurations with and without them, would clarify their contribution relative to retrieval alone.

5. **Replication with other models** Repeating the benchmark with other language models and retrieval engines would test whether the observed patterns are model-specific.

References

[AAA+23] Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[AK17] Bashair Althani and Souheil Khaddaj. Systematic review of legacy system migration. In *2017 16th International Symposium on Distributed Computing and Applications to Business, Engineering and Science (DCABES)*, pages 154–157. IEEE, 2017.

[Ann25] Lingareddy Annela. Ai-augmented legacy modernization: Transforming enterprise systems with smart automation. *Journal of Computer Science and Technology Studies*, 7(5):119–128, 2025.

[BLW+97] Jesus Bisbal, Deirdre Lawless, Bing Wu, Jane Grimson, Vincent Wade, Ray Richardson, and Donie O'Sullivan. An overview of legacy information system migration. In *Proceedings of Joint 4th International Computer Science Conference and 4th Asia Pacific Software Engineering Conference*, pages 529–530. IEEE, 1997.

[BMR+20] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

[CGeA24] Diogo Cosme, António Galvão, and Fernando Brito e Abreu. A systematic literature review on llm-based information retrieval: The issue of contents classification. *KDIR*, pages 135–146, 2024.

[Cha25] Soumen Chakraborty. Beyond etl: How ai agents are building self-healing data pipelines. *Journal of Computer Science and Technology Studies*, 7(3):741–756, 2025.

[CHGD24] Zina Chkirbene, Ridha Hamila, Ala Gouissem, and Unal Devrim. Large language models (llm) in industry: A survey of applications, challenges, and trends. In *2024 IEEE 21st International Conference on Smart Communities: Improving Quality of Life using AI, Robotics and IoT (HONET)*, pages 229–234. IEEE, 2024.

[CTS+24] Florin Cuconasu, Giovanni Trappolini, Federico Siciliano, Simone Filice, Cesare Campagnano, Yoelle Maarek, Nicola Tonellotto, and Fabrizio Silvestri. The power of noise: Redefining retrieval for rag systems. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 719–729, 2024.

[FAA25] Lucas Fernando Fávero, Nathalia Rodrigues de Almeida, and Frank José Affonso. A systematic mapping study on the modernization of legacy systems to microservice architecture. *Applied System Innovation*, 8(4), 2025.

[JLF+23] Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM Comput. Surv.*, 55(12), March 2023.

[Kou25] Paraskevas Koukaras. Data integration and storage strategies in heterogeneous analytical systems: Architectures, methods, and interoperability challenges. *Information*, 16(11), 2025.

[Kri25] Naveen Krishnan. Ai agents: Evolution, architecture, and real-world applications. *arXiv preprint arXiv:2503.12687*, 2025.

[LPP+20] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33:9459–9474, 2020.

[MAR25] Seyed Mahed Mousavi, Simone Alghisi, and Giuseppe Riccardi. Llms as repositories of factual knowledge: Limitations and solutions. *arXiv preprint arXiv:2501.12774*, 2025.

[MNY+26] Saroj Mishra, Suman Niroula, Umesh Yadav, Dilip Thakur, Srijan Gyawali, and Shiva Gaire. Sok: Agentic retrieval-augmented generation (rag): Taxonomy, architectures, evaluation, and research directions. *arXiv preprint arXiv:2603.07379*, 2026.

[NB25] Fnu Neha and Deepshikha Bhati. Traditional rag vs. agentic rag: A comparative study of retrieval-augmented systems. *Authorea Preprints*, 2025.

[OOJ+23] Olufunmilayo Ogunwole, Ekene Cynthia Onukwulu, Micah Oghale Joel, Ejuma Martha Adaga, and Augustine Ifeanyi Ibeh. Modernizing legacy systems: A scalable approach to next-generation data architectures and seamless integration. *International Journal of Multidisciplinary Research and Growth Evaluation*, 4(1):901–909, 2023.

[PE23] Sivakumar Ponnusamy and Dinesh Eswararaj. Navigating the modernization of legacy applications and data: Effective strategies and best practices. *Asian Journal of Research in Computer Science*, 16(4):239–256, 2023.

[PVN+24] Marcel Parciak, Brecht Vandevoort, Frank Neven, Liesbet M Peeters, and Stijn Vansummeren. Schema matching with large language models: an experimental study. *arXiv preprint arXiv:2407.11852*, 2024.

[PVN+25] Marcel Parciak, Brecht Vandevoort, Frank Neven, Liesbet M Peeters, and Stijn Vansummeren. Llm-matcher: A name-based schema matching tool using large language models. In *Companion of the 2025 International Conference on Management of Data*, pages 203–206, 2025.

[RB01] Erhard Rahm and Philip A Bernstein. A survey of approaches to automatic schema matching. *the VLDB Journal*, 10(4):334–350, 2001.

[SAKS+26] Hina Sattar, MA Al-Khasawneh, Umar Farooq Shafi, Touseef Hussain, Abdul Khader Jilani Saudagar, Syed Zain Ul Abideen, Mian Muhammad Kamal, and Said Baz Jahfar Khan. A data migration and integration approach for big data management using linked data. *Scientific Reports*, 2026.

[Sar22] Iqbal H Sarker. Ai-based modeling: techniques, applications and research issues towards automation, intelligent and smart systems. *SN computer science*, 3(2):158, 2022.

[SEK+25] Aditi Singh, Abul Ehtesham, Saket Kumar, Tala Talaei Khoei, and Athanasios V Vasilakos. Agentic retrieval-augmented generation: A survey on agentic rag. *arXiv preprint arXiv:2501.09136*, 2025.

[SM21] S Sivabalan and RI Minu. Heterogeneous data integration with elt and analytical mpp database for data analysis application. In *2021 Innovations in Power and Advanced Computing Technologies (i-PACT)*, pages 1–5. IEEE, 2021.

[SMOB25] Dachuan Shi, Olga Meyer, Michael Oberle, and Thomas Bauernhansl. Dual data mapping with fine-tuned large language models and asset administration shells toward interoperable knowledge representation. *Robotics and Computer-Integrated Manufacturing*, 91:102837, 2025.

[SPC+21] Kurt Shuster, Spencer Poff, Moya Chen, Douwe Kiela, and Jason Weston. Retrieval augmentation reduces hallucination in conversation. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 3784–3803, 2021.

[SRK25] Ranjan Sapkota, Konstantinos I Roumeliotis, and Manoj Karkee. Ai agents vs. agentic ai: A conceptual taxonomy, applications and challenges. *Information Fusion*, page 103599, 2025.

[Ven26] Padmanabhan Venkiteela. An enterprise agentic architecture framework for agentic ai governance and scalable autonomy. *Scientific Journal of Computer Science*, 2(1):1–17, 2026.

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[Wan17] Lidong Wang. Heterogeneous data and big data analytics. In *ACIS*, volume 3, pages 8–15, 2017.