Universiteit Leiden

Master Computer Science

Schema-Grounded LLM Framework for Multi-Source Enterprise QA in Nutrition Compliance

Name: Sha Li

Student ID: s4294092

Date: 24/06/2026

Specialisation: Artificial Intelligence

1st supervisor: Suzan Verberne

2nd supervisor: Zhaochun Ren

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden

The Netherlands

Abstract

Nutrition scientists and product developers often need to assess product nutrition performance, support product reformulation, and check nutrition or health claim conditions across markets. However, the required information is usually separated across internal Power BI dashboards, structured product data, and regional regulation resources, which makes these tasks difficult to answer in a direct and traceable way. In this thesis, we design and develop a multi-source visual analytics question-answering system for enterprise nutrition and health compliance. The system provides a unified natural-language interface for nutrition scientists and product developers to query internal Microsoft *Power BI* dashboards, open-source CSV data, and multi-region nutrition regulation resources.

Existing nutrition QA, compliance QA, and regulation retrieval studies mostly focus on consumer or clinical nutrition scenarios, or on regulatory texts themselves. They do not directly address executable multi-source QA over internal analytical data, structured regulation rules, and tool-backed analysis workflows.

To support this system, we propose an LLM orchestration framework for executable and traceable multi-source QA. Each query is first represented as a structured intent, including the task type, candidate sources, execution mode, and key slots needed for execution. The dual-layer schema grounds this intent by defining which sources, fields, metrics, and query patterns the system can use. Dynamic few-shot retrieval provides query-specific examples for intent parsing. For regulation-related questions, a structured regulation knowledge base supports rule retrieval, slot-aware reranking, and deterministic compliance calculation.

To evaluate the framework, we use CLINC150 as a public benchmark for intent parsing and construct an internal 120-question enterprise QA dataset covering representative dashboard, regulation, and compliance workflows. On the internal dataset, we evaluate structured intent parsing, regulation retrieval, and end-to-end executable QA. The results show that dynamic retrieval improves structured intent parsing, especially source routing, slot parsing, and core structured matching, and that these gains transfer to downstream executability and answer correctness.

Overall, the results suggest that enterprise nutrition compliance QA is better modeled as expert-oriented executable structured QA than as general free-form question answering. The framework connects natural-language business questions to real enterprise data, structured regulation evidence, and traceable execution steps, providing an extensible design for enterprise nutrition and health compliance analysis.

Contents

1 Introduction **1**

2 Related work **3**
- 2.1 Nutrition question answering and LLM-based nutrition support . . . 3
- 2.2 Compliance and regulation-oriented question answering 4
- 2.3 Grounded and executable question answering with retrieval, structure, and tools . . . 6

3 Problem setting **8**
- 3.1 Enterprise context and user needs . . . 8
- 3.2 Multi-source data landscape . . . 9
- 3.3 Task formulation for executable enterprise QA . . . 9
- 3.4 Structured intent representation . . . 10

4 Methodology **11**
- 4.1 Framework overview . . . 11
- 4.2 Dual-layer schema design . . . 12
- 4.3 Structured intent parsing . . . 12
- 4.4 Dynamic few-shot retrieval . . . 15
- 4.5 Intent-driven routing . . . 16
- 4.6 Constrained LLM orchestration . . . 16
- 4.7 Tool-backed execution and answer synthesis . . . 17
  - 4.7.1 Dashboard and structured data execution . . . 17
  - 4.7.2 Regulation retrieval . . . 18
  - 4.7.3 Compliance calculation . . . 19
  - 4.7.4 Execution of orchestrated plans . . . 19
  - 4.7.5 Answer synthesis . . . 20

5 Resources and datasets **20**
- 5.1 System resources and source schemas . . . 21
- 5.2 Construction of the structured regulation resource . . . 23
- 5.3 Public benchmark dataset: CLINC150 . . . 24
- 5.4 Construction of the enterprise 120-QA dataset . . . 27
- 5.5 Annotation structure and evaluation targets . . . 29

6 Experimental setup **30**

6.1 Overview of the experimental design 30
6.2 Experiment 1: Public benchmark validation on CLINC150 31
6.2.1 Objectives 31
6.2.2 Data 31
6.2.3 Compared methods 31
6.2.4 Evaluation metrics 32
6.2.5 Experimental protocol 33
6.3 Experiment 2: Structured intent parsing on the enterprise dataset 33
6.3.1 Objectives 33
6.3.2 Data 34
6.3.3 Compared methods 34
6.3.4 Evaluation metrics 35
6.3.5 Experimental protocol 36
6.4 Experiment 3: Regulation retrieval evaluation 37
6.4.1 Objectives 37
6.4.2 Data 37
6.4.3 Compared methods 37
6.4.4 Evaluation metrics 38
6.4.5 Experimental protocol 39
6.5 Experiment 4: End-to-end executable QA evaluation . 40
6.5.1 Objectives 40
6.5.2 Data 40
6.5.3 Compared methods 41
6.5.4 Evaluation metrics 41
6.5.5 Experimental protocol 42

7 Results **43**
7.1 Experiment 1: Public benchmark results on CLINC150 43
7.1.1 Main results 43
7.1.2 Analysis 43
7.1.3 Summary 44
7.2 Experiment 2: Enterprise structured intent parsing results 45
7.2.1 Main results 45
7.2.2 Analysis 45
7.2.3 Summary 48
7.3 Experiment 3: Regulation retrieval results 48

7.3.1 Main results 48
7.3.2 Analysis 49
7.3.3 Summary 50
7.4 Experiment 4: End-to-end pipeline results 50
7.4.1 Main results 50
7.4.2 Analysis 51
7.4.3 Summary 51

8 System implementation **52**
8.1 Prototype overview 52
8.2 Frontend implementation 53
8.3 Backend implementation 54

9 Discussion **55**
9.1 Overall implication of the results 55
9.2 Methodological implications 55
9.3 Practical implications 56
9.4 Limitations and trade-offs 56

10 Conclusion **57**

Appendix **59**
A. User interview summary 59
B. Enterprise dataset and annotation examples 60
C. Prompt templates 61
D. Schema examples 64
E. Anonymized system output examples 66

1 Introduction

In enterprise scenarios, users often need to answer business questions by consulting multiple data sources and tools. Natural language interfaces for business intelligence and structured data access address a related problem: how to connect user questions to relevant tables, views and queryable data sources [1, 2]. In the enterprise nutrition and health compliance setting studied in our work, the task goes beyond connecting a question to one structured source. Users may need to combine internal product metrics, dashboard results, and regional regulation evidence into a workflow that produces a traceable answer. As a result, even simple business questions may require users to switch between several systems and manually combine the results.

For example, exploratory interviews with nutrition scientists and collected dashboard-oriented business questions suggest that a nutrition scientist may want to compare nutrient values across products and markets, or check whether a product meets the condition for a certain nutrition claim under regional regulations. A product developer may want to quickly understand both product data and the related compliance requirements. These examples show why the system must connect natural-language questions to both analytical data and regulatory evidence.

Recent nutrition-related LLM studies show that natural language systems can support clinical nutrition, food product explanation, personalized nutrition reasoning, and structured food or nutrition tasks [3, 4, 5, 6]. However, these studies do not directly address enterprise nutrition compliance workflows for expert users. In our work, these users are nutrition scientists and product developers who need to analyze product nutrition performance, support reformulation decisions, and check market-specific nutrition claim conditions. Their questions often require internal dashboards, product metrics, and regulation rules to be used together.

Recent work on retrieval-augmented generation and tool-augmented language models shows that LLM-based systems can combine language generation with external knowledge access or task-specific actions [7, 8]. However, in the enterprise nutrition compliance setting, a useful system must do more than answer questions in a conversational way. It must understand the user query, identify the right data source, select the right tool, organize the execution workflow, and return a traceable result. The problem therefore involves source identification, tool selection, and controlled execution, not only language understanding.

This thesis studies how to build a schema-grounded LLM framework for executable enterprise question answering in nutrition and health compliance. The goal is to turn natural language questions into structured and traceable multi-source analysis workflows. In our setting, different questions require different execution paths: dashboard-related questions may need API-based query execution, local data questions may require CSV querying, and compliance-related questions may require regulation retrieval and eligibility calculation. In regulated-domain QA, systems need reliable evidence retrieval and rule matching, as shown in recent work on reg-

ulatory compliance NLP, regulatory question answering, and pharmaceutical regulatory RAG [9, 10, 11]. Therefore, the system must support intent-driven routing and constrained execution rather than simple answer generation.

The central idea of this thesis is to treat enterprise question answering as a form of executable structured question answering. Instead of asking the LLM to directly produce the final answer in free text, we first map the user query into a structured intent. This intermediate representation describes the task type, the analysis operation, the candidate data sources, and the key slots required for execution. Based on this structured intent, the system selects the appropriate source, chooses the correct tool path, and organizes either a simple single-source workflow or a complex multi-step workflow. This design makes the overall process more controllable, more traceable, and more suitable for enterprise use.

We therefore evaluate the role of schema grounding, dynamic few-shot retrieval, and constrained LLM orchestration in a multi-source enterprise setting. Prior work shows that in-context learning performance depends on the selected examples, and that retrieval-based example selection can help when the task has many labels or a large decision space [12, 13]. The main focus of this thesis is not pure text generation, but how natural language questions can be translated into executable workflows that access real sources and produce grounded results. We also study how improvements in structured intent parsing affect downstream executability and end-to-end answer correctness.

Based on the problem above, this thesis studies the following research questions:

RQ1. How can natural language queries in enterprise nutrition and health compliance be represented as structured intents that support data source identification, tool selection, and executable workflow construction?

This question focuses on task formulation and intermediate representation. We study how to map user questions into a structured form with question category, abstract operation, candidate data sources, and key slots, so that routing and execution become possible.

RQ2. Under schema constraints, to what extent can schema-grounded prompting and dynamic few-shot retrieval improve structured intent parsing and downstream executability?

This question focuses on the main method of this thesis. We study how different prompting and retrieval strategies affect category prediction, operation prediction, routing, slot extraction, and core structured matching, and whether these improvements transfer to downstream execution.

RQ3. To what extent can the proposed framework support correct and traceable end-to-end execution for dashboard analytics, structured data queries, and regulation-related tasks?

This question focuses on the overall effectiveness of the framework, including regulation retrieval, end-to-end answer generation from executed workflows, and practical usefulness in a real enterprise setting.

The main contributions of this work are as follows:

- We propose a schema-grounded LLM orchestration framework for executable enterprise question answering in nutrition and health compliance.
- We formalize enterprise natural language analysis questions as a structured intent representation problem, so that the system can perform intent-driven routing, tool selection, and constrained execution based on question category, abstract operation, candidate data sources, and slot information.
- We construct a 120-question annotated enterprise evaluation dataset derived from internal dashboard-oriented business questions for evaluating structured intent parsing, regulation retrieval, and end-to-end executable question answering.
- We build a structured multi-region nutrition regulation resource to support regulation retrieval and compliance-related analysis.
- We conduct a multi-level evaluation, including public benchmark validation, full-dataset intent parsing experiments, regulation retrieval experiments, and end-to-end system evaluation.
- We develop a prototype enterprise question answering system with an extensible architecture that supports multiple data sources, tool routing, and the integration of dashboard analytics, structured data querying, regulation retrieval, and natural language interaction.

2 Related work

Direct prior work on enterprise-facing nutrition compliance question answering is still limited, especially work that combines nutrition regulations, internal dashboards, product-level metrics, and executable workflows. We therefore position our work with respect to three adjacent lines of research.

The first line is nutrition question answering and LLM-based nutrition support. The second line is compliance and regulation-oriented question answering. The third line is grounded and executable question answering with retrieval, structure, and tools. Together, these lines of work provide the context for the gap addressed in our work.

2.1 Nutrition question answering and LLM-based nutrition support

AI and LLMs have already been studied in clinical nutrition and nutrition support. Belkhouribchia and Pen summarize uses such as dietary recommendation, nutrition education, nutrient analysis, data extraction, and clinical decision support, and they also highlight risks such as factual errors, hallucination, limited domain specificity, and the need for human oversight [3]. Varayil et al. also discuss AI for

clinical nutrition, including outcome prediction, risk identification, decision support, and personalized education, while noting that accuracy, accessibility, and ethical issues still need further evaluation [14]. These studies provide the nutrition-domain background for our work, but they mainly focus on clinical or patient-support settings.

Beyond these broader clinical-nutrition studies, several works study nutrition assistants and nutrition QA tasks more directly. Szymanski et al. develop a food product nutrition assistant and use feedback from registered dietitians to refine prompt templates for food product explanations [4]. Zhang et al. introduce NGQA, a nutritional graph question answering benchmark for personalized health-aware reasoning, where models judge whether a food is healthy for a specific user and explain the role of relevant nutrients [5]. Gjorgjevikj et al. introduce FoodyLLM for structured food and nutrition tasks, including recipe nutrient estimation, traffic-light classification, and ontology-based entity linking [6]. Although these studies focus on different nutrition-related tasks, they all connect natural language input with nutrition-domain knowledge to answer, explain, or judge questions about food, nutrients, and nutrition labels. They show that nutrition QA is not only a retrieval and text generation problem, but also requires domain knowledge, structured nutrition information, and explainable outputs.

However, their setting is different from ours. Most of these systems support consumers, patients, dietitians, or clinicians. In this thesis, the users are enterprise nutrition scientists and product developers. Their questions involve product metrics, market filters, nutrition claims, regulatory thresholds, Power BI dashboards, and internal data sources. This leaves a gap for expert-oriented enterprise nutrition compliance QA, where natural language questions must be connected to structured intents, source routing, dashboard execution, regulation retrieval, and traceable answer generation.

2.2 Compliance and regulation-oriented question answering

A second relevant line of work is compliance and regulation-oriented question answering. Although this work usually does not focus on nutrition compliance, it addresses similar regulated-domain QA problems. Existing work can be grouped in two main directions. The first direction is regulatory QA and regulatory RAG, where systems retrieve relevant rules from complex regulatory texts and generate grounded answers. The second direction is automated compliance checking, where systems represent regulatory requirements as structured conditions and check whether a target document, system, or requirement satisfies them.

Jain et al. review the use of AI and NLP for regulatory compliance [9]. They show that NLP and LLMs can support compliance document analysis, regulation accessibility, question answering, and regulatory change support. They also point out challenges such as adaptability, completeness, trust, the lack of standardized benchmarks, and the need for human-AI collaboration. This shows that compliance-oriented QA requires more than text understanding. It also needs reliable evidence,

clear explanations, and auditable decisions.

In regulatory QA and regulatory RAG, RAGulator formulates the task as passage retrieval followed by answer generation: the system first retrieves relevant obligations and rules from regulatory texts, and then generates answers based on the retrieved passages [10]. Work on legal information retrieval and entailment also shows that legal tasks require accurate matching between questions and legal provisions, using lexical retrieval and transformer-based representations [15]. QA-RAG studies a similar problem in pharmaceutical regulatory compliance. It shows that domain-specific regulatory texts can be hard to retrieve with the user query alone, because the wording of the query and the wording of the guideline may differ [11]. These studies show that compliance-related QA has high requirements for evidence retrieval and rule matching.

In automated compliance checking, recent studies represent regulatory requirements in more structured or computable forms. Sun et al. propose a RAG-based compliance checking framework for checking whether business processes comply with laws, regulations, and industry standards. Their work combines structured knowledge, regulatory and business process information, retrieval, and reasoning [16]. Kruiper et al. propose a platform-based strategy for digitalising regulatory compliance processes in the built environment. They show that extracting relevant requirements from large regulatory document collections is central to compliance checking, and that ambiguity, inconsistent terminology, contradictory requirements, and multi-word expressions remain important challenges [17]. Falcarin et al. convert regulatory texts and software requirements specifications into knowledge graphs, and compare the graph representations to support auditor-oriented compliance assessment [18]. These studies show that compliance decisions often require rules to be represented as retrievable, checkable, and traceable conditions, not only as natural language explanations.

Overall, these studies are relevant because they address common problems in regulation-oriented QA and automated compliance checking. Systems need to retrieve reliable evidence from regulatory or compliance documents, match the applicable rules accurately, represent requirements in checkable forms, and support decisions in an explainable and auditable way. The enterprise nutrition compliance setting shares these requirements, because retrieving the wrong rule, misrepresenting a requirement, or interpreting a rule incorrectly can affect the downstream compliance decision.

The difference is that our setting also requires a connection to enterprise analytical data. Some questions require product or portfolio metrics from internal compliance dashboards. Other questions require structured claim rules from a claim-rule knowledge base and an eligibility decision based on nutrient values. Most existing regulatory QA and compliance checking work focuses on regulatory texts, privacy policies, pharmaceutical guidelines, legal provisions, or software requirements themselves. It pays less attention to how regulatory rules can be connected to enterprise analytical data. Our work focuses on this connection between regulation-oriented

QA, structured claim rules, and enterprise nutrition analytics.

2.3 Grounded and executable question answering with retrieval, structure, and tools

A third relevant line of work is grounded and executable question answering. This section reviews methods for turning natural language questions into executable and traceable workflows. The discussion focuses on structured intent representation, schema-aware data access, retrieval for examples and rules, and tool-backed execution with executable evaluation.

Structured representations are commonly used to connect user utterances with downstream systems. In task-oriented NLU, intent detection identifies the user's goal, while slot filling extracts semantic information needed to fulfil that goal [19]. This is relevant because enterprise QA also needs an intermediate representation before execution. In a multi-source setting with several dashboards, structured tables, and regulation resources, the system must normalize the user request before it can select a source, apply a schema, or call a tool. Therefore, the representation needed here is more execution-oriented than traditional intent-slot frames: it must support decisions about data sources, operations, execution conditions, and workflow selection.

Dynamic in-context learning provides a way to improve this type of structured prediction without fine-tuning. Rubin et al. show that retrieving suitable demonstrations can improve in-context learning [12], and Milios et al. study retrieval-based in-context learning for text classification with many labels [13]. These works are relevant because retrieval can help select examples that are closer to the current query, which can make structured parsing more stable.

Schema awareness is central to executable QA over structured data. Text-to-SQL studies how natural language questions can be mapped to executable SQL queries. Liu et al. summarize LLM-based Text-to-SQL work and highlight challenges such as natural language ambiguity, under-specification, and mapping natural language to database schemas and instances. They also describe modern Text-to-SQL systems as modular pipelines with components such as schema linking, database content retrieval, intermediate representations, correction, and execution-guided refinement [2]. DIN-SQL follows a similar direction by decomposing Text-to-SQL into subtasks such as schema linking, query classification and decomposition, SQL generation, and self-correction [20]. These studies show that executable data access usually requires schema grounding and intermediate processing, rather than a single unconstrained generation step.

Business intelligence settings make schema grounding more difficult. ChatBI studies NL2BI as a practical production scenario of NL2SQL and shows that BI data often contains many columns and ambiguous column names, making LLM-based schema linking difficult under token limits. It addresses this problem by first selecting a single relevant view before passing the selected view columns to the LLM [1]. This is relevant to enterprise dashboard analytics, where the visible source and

schema space must be reduced before detailed field or metric grounding.

However, the execution environment in this thesis is different from the SQL setting studied in most Text-to-SQL and NL2BI work. The system interacts with Power BI semantic models and uses DAX for dashboard queries. DAX is the expression language used for measures, filters, and business calculations in Power BI semantic models,[1] and DAX queries can be executed against Power BI semantic models through the Execute Queries REST API.[2] Therefore, Text-to-SQL and NL2BI provide the closest research background, while Power BI and DAX define the technical execution context.

Retrieval has more than one role in grounded and executable QA. In RAG, retrieval provides external knowledge for knowledge-intensive generation [7]. In dynamic in-context learning, retrieval selects examples relevant to the current input [12, 13]. In regulation-oriented tasks, retrieval locates applicable rules or provisions. Methodologically, this connects to lexical retrieval, semantic retrieval, and reranking. BM25 is a standard lexical retrieval method for ranking documents based on term matching [21]. SBERT provides sentence-level semantic embeddings that can be compared with cosine similarity [22]. These methods are often combined in retrieval pipelines, where lexical retrieval supports candidate recall and semantic models support ranking or reranking. This is relevant to compliance-oriented QA because retrieving or ranking the wrong rule can affect the downstream decision.

Executable QA must also interact with external tools or data environments. ReAct shows that LLMs can interleave reasoning traces with task-specific actions and use these actions to interact with external knowledge sources or environments [8]. This is relevant because enterprise nutrition compliance QA cannot rely only on direct text generation. It must interact with internal dashboards, structured data sources, and regulation knowledge bases to obtain reliable metrics, rules, and evidence. However, enterprise compliance analysis differs from open-ended agent settings. External interaction needs to be constrained by schemas, source constraints, business rules, validation, deterministic calculation, and traceability.

Executable QA also requires evaluation based on structured or executed results. Spider defines a complex and cross-domain text-to-SQL task where models must map natural language questions to executable SQL queries and generalize to new database schemas [23]. DIN-SQL also reports execution accuracy and notes that exact match and execution accuracy can differ because more than one valid query may answer the same question [20]. These studies show that executable QA should not be evaluated only by surface text similarity. When the output can be a value, list, ranking, trend, regulation entry, or compliance decision, evaluation should focus on whether the structured or executed result is correct.

Overall, these studies provide the methodological background for our work. Existing work studies structured language understanding, example retrieval, schema-grounded data access, evidence retrieval, rule matching, tool use, and execution-

[1] `https://learn.microsoft.com/en-us/dax/dax-overview`

[2] `https://learn.microsoft.com/en-us/rest/api/power-bi/datasets/execute-queries`

based evaluation. However, these capabilities are usually studied separately. Intent-slot work mainly focuses on language understanding, Text-to-SQL and NL2BI mainly focus on structured data querying, RAG and legal retrieval mainly focus on evidence or rule retrieval, tool-use work mainly studies interaction with external environments, and Text-to-SQL evaluation mainly focuses on SQL queries or executed query results.

In contrast, enterprise nutrition compliance QA requires these capabilities to be combined in one controlled workflow. The system needs to convert natural language questions into executable structured intents, select relevant data or regulation sources under schema constraints, obtain reliable results through dashboard querying, rule retrieval, and deterministic compliance checking, and finally support traceable answer generation and operation-specific evaluation. Therefore, the gap addressed by our work is not a single new algorithm for one module. Instead, our work studies how these methods can be integrated into an executable, traceable, and multi-source QA framework for enterprise nutrition compliance, and how this combined framework can be evaluated for real analytical and compliance tasks.

3 Problem setting

3.1 Enterprise context and user needs

The problem studied in this thesis comes from real business needs in enterprise nutrition and health analysis. In this setting, scientific validity, regulatory accuracy, and decision transparency are all important. Users such as nutrition scientists and product developers often need to check product nutrition performance, compare indicators across markets or product groups, and decide whether a product meets the condition for a certain nutrition claim. These tasks are directly related to product analysis, compliance assessment, and business decision making.

However, the current analysis workflow is often inconvenient. Relevant information is usually distributed across several internal dashboards, structured data tables, and regulation sources. As a result, users often need to switch between multiple systems, align filters and metric definitions, and manually combine results before they can reach a conclusion. Even when the question itself is clear, the path to the answer is often fragmented and time-consuming.

To better understand these needs, we conducted three exploratory interviews with nutrition scientists from different regions in Unilever. These interviews were used to identify representative question types, common workflow pain points, and user expectations for a natural language system. They showed that user questions are usually not isolated information requests. Instead, they are closely connected to concrete analysis tasks and practical decisions. For example, users may want to compare nutrition indicators across products or markets, determine whether a product satisfies a regional nutrition claim condition, or combine information from several sources to support product development and compliance decisions.

These observations suggest one central requirement. Users do not simply need a chat system that produces fluent text answers. They need a system that can support their actual analysis workflow. Such a system should help them move from a natural language question to a usable analytical result in a more direct and reliable way. This requirement motivates the problem setting of this thesis.

3.2 Multi-source data landscape

The problem above takes place in a multi-source and heterogeneous data environment. To answer typical enterprise nutrition and health questions, the system must work with several different types of sources.

First, internal Power BI dashboards contain many key business indicators and support tasks such as real-time querying, comparison, ranking, trend analysis, and product detail analysis. These dashboards are designed for different business themes and expose different dimensions, measures, and filters. Second, some questions depend on local structured data or public external data, which are usually stored as tables or CSV files. These sources are suitable for filtering, aggregation, and focused analysis over well-defined fields. Third, regulation-related questions require access to regulation knowledge resources in order to support the retrieval and matching of claim conditions, threshold values, and regional differences.

These source types are heterogeneous in several ways. They differ in structure, field names, metric definitions, coverage, and access methods. Different dashboards focus on different analytical views. Different tabular sources use different granularities and field organizations. Regulation knowledge is rule-based, region-dependent, and expressed in a form that is different from standard analytical tables. Because of these differences, user questions cannot be mapped directly to one single table or one unified query interface.

This source heterogeneity is not only background information. It is one of the main conditions that shapes the task studied in this thesis. Since the system must work across different source types, it cannot stop at general question understanding. It must also determine which data environment is relevant to the question and support the later retrieval, querying, or calculation process in that environment.

3.3 Task formulation for executable enterprise QA

Based on the user needs and the multi-source setting above, we formulate the problem as a form of executable question answering for enterprise analysis. The input to the system is a natural language question. The output is not just a free-form text response, but a traceable analysis result supported by real data or regulation evidence.

Under this formulation, answering a question requires more than generating text. The system must first understand the task expressed in the question. It must then identify the relevant source or sources, determine the analytical action that is needed, and organize the corresponding execution process. The problem therefore combines

question understanding with executable analysis.

In this thesis, this formulation differs from traditional question answering in two main ways. First, the goal is not only answer generation, but also grounded execution over real enterprise sources. Second, the question understanding problem is not a single-label classification problem. A user question may contain information about task type, target source, analytical action, and execution conditions at the same time. For this reason, the system must support a layered interpretation of the question before execution becomes possible.

This formulation also reflects the fact that enterprise questions may require different execution styles. Some questions can be answered with one source and one fixed workflow. Others may require multiple sources, multiple steps, or the combination of analytical results with regulation evidence. Therefore, the task studied in this thesis can be viewed as a joint question understanding and execution problem in a multi-source enterprise environment. The main goal is not only to let the system answer questions, but to let it support the correct analysis process behind the question.

3.4 Structured intent representation

To support this formulation, we use a structured intent representation as an intermediate form between natural language input and later execution. The main role of this representation is to turn an open and ambiguous natural language question into a structured form that can support later system decisions.

In this thesis, the structured intent includes several types of information that are necessary for execution. These include the question category, the abstract operation, the candidate data sources, the execution mode, and the key slots required by later modules. The question category distinguishes broad task types such as data query, compliance check, regulation lookup, or other system-related questions. The abstract operation describes the main analytical action requested by the user. Candidate data sources indicate the possible source space relevant to the query. The execution mode indicates whether the query follows a simpler fixed workflow or a more complex multi-step path. The slots record the key parameters needed later, such as market, product, nutrient, time period, claim type, or regulation region.

This intermediate representation is important for two reasons. First, it gives the system a unified and explicit result after language understanding. This reduces the instability of moving directly from natural language to tool execution. Second, it provides a structured interface for later modules. Instead of relying on a free-form interpretation of the question at every later stage, the system can operate on a clearer and more constrained representation.

For this reason, structured intent representation is a key part of the problem setting in this thesis. It makes the task more precise, and it creates a bridge between question understanding and executable analysis. In the following chapter, we describe how this representation is used in the proposed framework to support routing, constrained orchestration, and tool-backed execution.

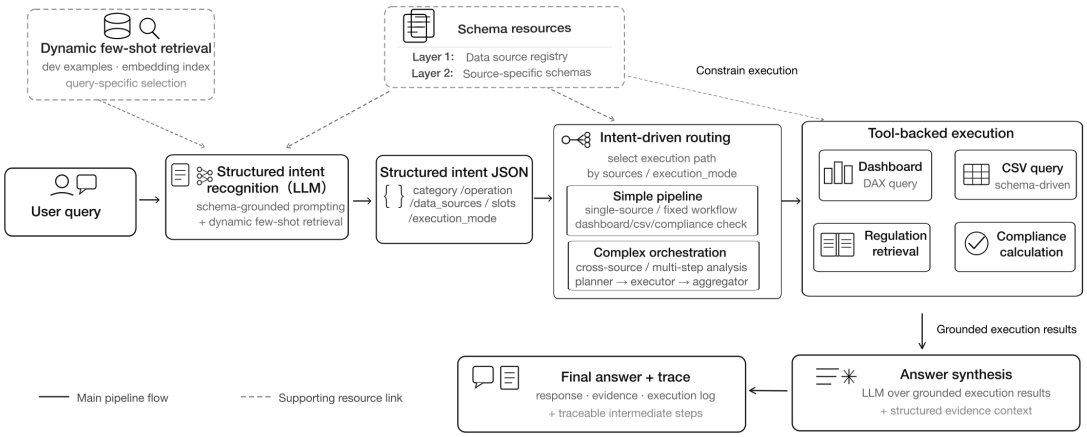


Figure 1: Overview of the proposed schema-grounded orchestration framework. The figure shows the main components of the pipeline, from structured intent recognition and routing to tool-backed execution and answer synthesis.

4 Methodology

4.1 Framework overview

The proposed framework adopts a unified-entry architecture for executable enterprise question answering. A user query first enters the system through a single natural language interface. The system then performs structured intent recognition, determines the appropriate execution path, executes source-specific operations, and produces a final answer together with trace information. This design supports different types of enterprise queries within one controlled workflow.

Figure 1 gives an overview of the proposed schema-grounded orchestration framework. It shows how dynamic few-shot retrieval and schema resources support structured intent recognition, how the parsed structured intent drives routing, and how source-specific execution modules produce grounded outputs for answer synthesis.

At a high level, the framework contains four main stages. The first stage is *structured intent recognition*. In this stage, the system converts the natural language query into a structured intent representation. This stage is supported by schema-grounded prompting and dynamic few-shot retrieval. The second stage is *intent-driven routing*. Based on the parsed intent, the system decides which execution path should be used. Depending on the query type and execution mode, the system may enter a simple pipeline for dashboard, CSV, compliance-related queries, or a complex orchestration pipeline for cross-source and multi-step analysis. The third stage is *tool-backed execution*. In this stage, the selected sources are queried through source-specific modules, such as dashboard DAX querying, CSV querying, regulation retrieval, and compliance calculation. The fourth stage is *answer synthesis*. In this stage, the system organizes the grounded execution outputs into a final natural language response and records step-level trace information.

This framework therefore does not treat all queries as one fixed pipeline. In-

stead, it uses structured intent recognition to determine which execution branch is appropriate for the current query. Simple queries follow a direct execution path or a fixed workflow. More complex queries enter an orchestration path that decomposes the task into several executable steps through a planner, executor, and aggregator. All paths finally return a traceable answer grounded in retrieved data or regulation evidence.

4.2 Dual-layer schema design

To support stable routing and executable analysis, the framework uses a dual-layer schema design. The main goal of this design is to constrain both the searchable source space and the executable action space of the system.

The first layer is a *global data source registry*. This layer stores high-level metadata about the available sources in the system. It includes information such as source type, source identifier, description, signature metrics, supported capabilities, and possible join keys. Its main role is to support registry-based source discovery and high-level routing. For example, the system uses this layer to distinguish between Power BI dashboards, local CSV-based sources, and regulation resources before later execution starts.

The second layer is a set of *source-specific schemas*. Each source has its own schema describing the internal executable structure of that source. Depending on the source type, this schema may contain dimensions, attributes, slots, measures, metric groups, output constraints, and execution-specific metadata. These schemas are used later for slot normalization, metric validation, field mapping, query generation, and execution control. In this way, the system does not rely on unconstrained free-form guessing of fields or operations. Instead, later execution is grounded in source-specific structure.

The two layers serve different but connected roles. The global registry narrows the source space and supports high-level routing decisions. The local source schema supports source-level interpretation and execution. Together, they make the framework operational rather than purely descriptive. The registry helps determine where a query should go, while the local schema helps determine how that query should be executed once a source has been selected.

This design is especially useful in a heterogeneous enterprise environment. The framework must work across dashboards, structured tables, and regulation resources, but these source types do not share one single flat schema. The dual-layer design provides a practical compromise. It offers a unified system-level source view while preserving the execution-specific structure of each individual source.

4.3 Structured intent parsing

On top of the dual-layer schema design, the framework performs structured intent parsing. This module converts the natural language query into a structured representation that can directly support routing and later execution.

Table 1: Main components of the structured intent representation.

Component	Value space / Examples	Main Role
Question category	`data_query`, `compliance_check`, `regulation_lookup`, `meta`	task typing
Abstract operation	`query_value`, `rank_by_dimension`, `list_item`, `compare_values`, `trend_over_time`, `get_product_detail`, `check_eligibility`, `get_regulation_standard`, `other`	analytical action
Data sources	Source identifiers from the global registry, such as `usnc_pns`, `hfss_6npm`, `ns_6npm`, or `regulations`	source routing
Execution mode	`simple`, `complex`	execution style
Slots	Schema-defined execution parameters, such as `market`, `period`, `claim_type`, `nutrient`, or `regulation_region`	execution parameters

In this thesis, the formal structured intent contains five core fields:

- `question_category`: the broad task type of the query
- `abstract_operation`: the main analytical action requested by the user
- `data_sources`: the candidate sources relevant to the query
- `execution_mode`: the expected execution style of the query
- `slots`: the key execution parameters extracted from the query

These core fields form the main intermediate representation between natural language understanding and later execution. In implementation, the system also uses several auxiliary control fields, such as dashboard-only subsets, regulation requirements, orchestration flags, grouping fields, ordering fields, and output-field constraints. These auxiliary fields are not part of the core formal representation itself, but they help operationalize the final execution path.

The framework uses four question categories: `data_query`, `compliance_check`, `regulation_lookup`, and `meta`. These categories distinguish the broad goal of the query and strongly affect the later pipeline decision. The framework also uses nine abstract operations: `query_value`, `rank_by_dimension`, `list_item`, `compare_values`, `trend_over_time`, `get_product_detail`, `check_eligibility`, `get_regulation_standard`, and `other`. The abstract operations cover scalar lookup, ranking, listing, comparison, trend analysis, product-detail lookup, eligibility checking, regulation lookup, and out-of-scope handling. These operations define the analytical action more precisely than the question category alone. Finally, the framework distinguishes two execution modes: `simple` and `complex`. The simple mode is used for queries handled through a direct execution path. The complex mode is used for queries that require orchestration, such as cross-source analysis or multi-step reasoning over dashboard outputs.

Table 1 summarizes these main intent components.

This design means that intent parsing in the framework is not a single-label classification problem. Instead, it is a structured prediction problem. A single enterprise query may simultaneously contain information about task type, target source, analytical action, execution complexity, and key filters. The parser therefore produces a structured JSON object rather than a single output label.

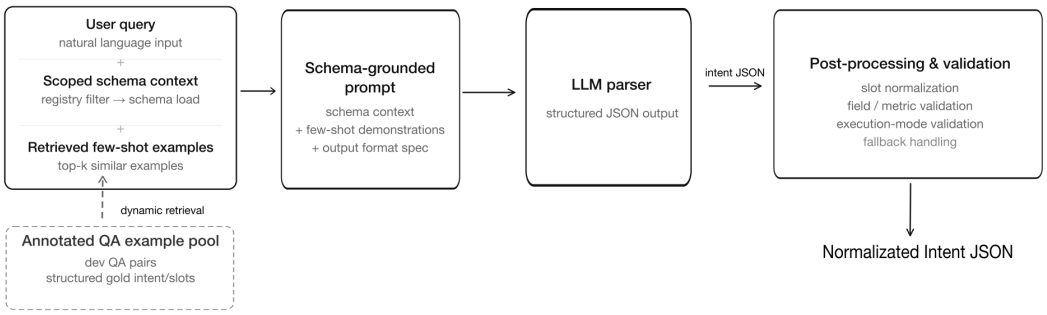


Figure 2: Structured intent parsing process. The figure shows how the user query, scoped schema context, and retrieved examples are combined to produce a normalized structured intent.

Figure 2 summarizes the structured intent parsing process. It shows how the user query, scoped schema context, and retrieved few-shot examples are combined into a schema-grounded prompt, and how the generated intent JSON is normalized and validated before routing and execution.

To support this process efficiently, the framework uses registry-based candidate filtering before prompt construction. The query is first matched against source-level registry metadata such as signature metrics and source descriptions. Based on this result, the system loads only the most relevant source context into the prompt instead of injecting all available schemas at once. This reduces prompt length and narrows the interpretation space.

The parser then uses schema-grounded prompting to generate the initial structured intent. The prompt includes source-relevant schema context, slot and measure information, source-specific constraints, and the output format specification. The model returns the result as structured JSON. After that, the framework applies a post-processing stage to validate and refine the output. This stage checks field consistency, normalizes slots, validates metrics, derives Power BI dashboard subsets from the general `data_sources` field, and adds auxiliary execution-control information when needed. In particular, slot normalization maps extracted user expressions to the source-specific schema. Time expressions are aligned with the period format used by the selected source, while values such as market, region, brand, product group, claim type, and nutrient are matched against schema-defined enumerable values or allowed field values. This makes the structured intent directly usable for routing and executable query construction.

A simplified example is shown below.

Example: Structured intent parsing

Question: *What is the USNC compliance rate in Netherlands in 2025 Q3?*

Parsed intent:

```
{
```

```
  "question_category": "data_query",
  "abstract_operation": "query_value",
  "data_sources": ["usnc_pns"],
  "execution_mode": "simple",
  "slots": {
    "period": "2025 Q3",
    "market": "Netherlands",
    "region": null,
    "brand": null,
    "product_group": null,
    "nutrient_values": []
  }
}
```


This structured intent becomes the first executable decision layer of the framework. Later modules do not need to reinterpret the raw query from scratch. Instead, they can operate on a constrained and validated intermediate representation.

4.4 Dynamic few-shot retrieval

In the final framework, structured intent parsing is implemented with schema-grounded prompting plus dynamic few-shot retrieval. This follows prior work showing that retrieved demonstrations can improve in-context learning by providing query-specific examples [12, 13]. As shown in Figure 2, the system does not use the same fixed demonstrations for every query. Instead, it retrieves query-specific labeled examples from the development example pool and injects them into the prompt as in-context demonstrations.

The motivation for this design is that enterprise nutrition and health queries belong to a limited task space, but their surface forms vary considerably. Different users may ask the same analytical question with different business terms, different levels of detail, or different slot combinations. Under these conditions, a fixed demonstration set may be poorly matched to the current query. Dynamic retrieval makes the demonstrations more relevant to the input query and helps the parser better infer the intended category, operation, source signals, and slots.

In implementation, the retrieval pool is built from labeled development examples. Each example query is encoded into an embedding representation, and the system builds a retrieval index over these development queries. For a new input query, the system encodes the query, retrieves the top-k most similar development examples, and injects the retrieved examples into the schema-grounded prompt. The retrieved examples include the structured intent labels used by the parser, so they act as query-specific demonstrations for the same structured prediction task.

More formally, for an input query q and a development query q_i, the system encodes them into embedding vectors $e(q)$ and $e(q_i)$. It then ranks development examples by cosine similarity:

$$s(q, q_i) = \frac{e(q) \cdot e(q_i)}{\|e(q)\| \|e(q_i)\|}.$$

The top-k examples according to $s(q, q_i)$ are selected as the dynamic few-shot demonstrations.

This design is not only used to improve upstream parsing quality. Since the parsed intent drives routing and later execution, better demonstrations can also improve downstream executability. For this reason, dynamic few-shot retrieval is part of the final system design rather than only an experimental variation. Later experiments compare this design against zero-shot and static few-shot alternatives, but the final framework uses the dynamic retrieval setting as the operational intent parsing configuration.

4.5 Intent-driven routing

After structured intent parsing, the framework performs intent-driven routing. The goal of this stage is to determine which execution path should be used for the current query.

Routing in the framework is performed in two layers. The first layer is *pipeline-level routing*. At this level, the system determines the overall query path based on the parsed intent and auxiliary control fields. Depending on the query, the system may choose a dashboard query pipeline, a regulation-only pipeline, a combined dashboard-plus-regulation pipeline, a CSV-based analysis pipeline, or a dashboard orchestration pipeline for complex queries. This decision is mainly driven by deterministic control logic over the parsed intent, rather than by free-form LLM routing.

The second layer is *dashboard-level routing and validation*. This layer is only relevant for dashboard-oriented queries. In this stage, the framework validates the dashboard candidates derived from the intent and, when necessary, applies a rule-based fallback using registry information. For multi-dashboard queries, it also determines join keys or aggregation modes needed for later execution. In this sense, dashboard routing is not a second semantic parser. It is a validation and execution-preparation layer built on top of the structured intent.

This two-layer design separates high-level execution path selection from dashboard-specific execution validation. The first layer decides what kind of pipeline the query should enter. The second layer refines the dashboard path when dashboard execution is required. This makes routing more stable and easier to control than direct end-to-end source selection from raw text.

4.6 Constrained LLM orchestration

For complex dashboard analysis, the framework enters a constrained orchestration stage. This stage is used when a query cannot be answered by a single simple dashboard operation. Typical cases include cross-source analysis and other multi-step queries.

The word *constrained* is important here. The framework does not rely on an unrestricted agent planner that can generate arbitrary actions. Instead, it plans inside a bounded execution space defined by source schemas, available analysis types, supported operations, and merge strategies. This keeps the system more controllable and more suitable for enterprise use.

The orchestration layer uses a hybrid planning strategy. For known analysis patterns, the framework prefers deterministic or configuration-driven planning. For example, cross-source joins can be generated from predefined planning logic or configuration templates. For more complex semantic cases, such as multi-step analytical requests, the framework uses a schema-aware semantic planner. If no known planning pattern is matched, the framework can fall back to LLM-based planning as a last resort.

This means that orchestration in the framework is not based on one single planning mechanism. Instead, it is analysis-type-dependent. Known patterns are handled through faster and more stable planning strategies, while harder semantic cases are handled with LLM assistance under schema and execution constraints. Numerical calculations are never delegated to free-form LLM reasoning. They are handled later by deterministic program logic.

A typical orchestration output contains a bounded execution plan with fields such as analysis type, ordered execution steps, merge strategy, and post-processing requirements. Each step refers to a specific source, operation, slot set, and dependency relation. The plan is then executed by downstream source-specific tools without requiring the model to generate the final answer directly.

4.7 Tool-backed execution and answer synthesis

After routing or orchestration, the framework performs tool-backed execution. This stage executes the selected source operations and produces grounded outputs for final answer synthesis.

4.7.1 Dashboard and structured data execution

For dashboard-oriented queries, the system uses dashboard-specific execution modules. Given a normalized intent JSON, the query composer uses the parsed `data_sources` field to identify the target dashboard schema. It then constructs an executable DAX query by mapping the selected `abstract_operation`, normalized slots, target metrics, filter conditions, grouping fields, ordering fields, and ranking constraints to the corresponding tables, fields, and measures defined in the source schema. In the current framework, this is implemented as a Text-to-DAX step, where the generated query is expressed in DAX.[3] The query engine then executes the DAX query against the selected Power BI semantic model through the Execute Queries REST API.[4] The returned records are normalized into a structured result

[3] `https://learn.microsoft.com/en-us/dax/dax-overview`

[4] `https://learn.microsoft.com/en-us/rest/api/power-bi/datasets/execute-queries`

format for later answer synthesis.

The role of `abstract_operation` is especially important in this stage. It acts as one of the main control signals for query composition. Different operations correspond to different query-construction patterns. For example, `query_value` usually maps to scalar aggregation, `list_item` maps to row-level listing, `rank_by_dimension` maps to ranking queries, `compare_values` maps to grouped comparison patterns, and `trend_over_time` maps to time-based grouping patterns. In this way, the structured intent acts as the bridge between language understanding and executable dashboard querying.

For CSV-based structured data sources, the framework uses source-specific querying modules to perform filtering, aggregation, ranking, or comparison over local tabular data. These paths follow the same overall design principle: the execution is driven by parsed intent and schema constraints, not by unconstrained text generation. Although the execution interfaces differ across source types, the framework normalizes the outputs into a form that can be handled consistently in later answer synthesis.

4.7.2 Regulation retrieval

For regulation-related questions, the framework uses a dedicated regulation retrieval module over a structured regulation resource. Unlike general document retrieval, this module does not retrieve arbitrary text chunks from raw regulatory documents. Instead, it operates over normalized regulation entries that contain both textual evidence and structured rule fields, such as region, claim name, nutrient, threshold value, unit, basis, and citation.

Figure 3 summarizes the two-stage regulation retrieval process.

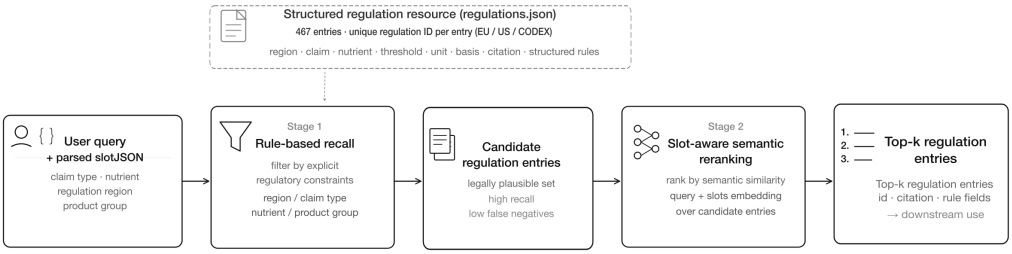


Figure 3: Regulation retrieval process. The figure shows the two-stage retrieval pipeline from rule-based recall to slot-aware semantic reranking.

The input to this module is the user query together with the regulation-related slots extracted during structured intent parsing. These slots may include claim type, nutrient, regulation region, product group, and food category. This slot information is important because nutrition claim rules are condition-dependent. The same claim may have different thresholds across regions, product forms, or product categories.

The retrieval process follows a two-stage design. In the first stage, the system performs rule-based recall. It uses structured slots such as claim type, nutrient,

region, and optional product group to construct a regulation-compatible candidate set under explicit rule constraints. In the second stage, the system performs slot-aware semantic reranking over the candidate entries recalled in the first stage, using sentence-level semantic representations following the SBERT approach [22]. The query representation combines the raw question with the relevant structured slots, and each candidate regulation entry is represented by its claim name, wording variants, threshold text, regulation text, and structured rule fields.

Let x_r denote the text-rule representation of a candidate regulation entry r. For each candidate entry $r \in C(q)$, the system computes a reranking score based on the cosine similarity between the slot-augmented query representation q_{slot} and x_r:

$$s(q, r) = \cos\Big(e(q_{\text{slot}}), e(x_r)\Big).$$

The candidates are sorted by $s(q, r)$, and the top-ranked entries are passed to compliance calculation or answer synthesis.

This design combines controllable recall with semantic refinement. Pure semantic retrieval may retrieve a rule that is textually similar but legally inapplicable, while purely rule-based filtering may be unstable when several related candidates remain. The hybrid design first restricts the candidate space with structured regulatory constraints, and then uses semantic similarity to select the most relevant entry within that candidate set.

4.7.3 Compliance calculation

For compliance-check questions, regulation retrieval is followed by a deterministic eligibility calculation stage. This module compares user-provided or dashboard-retrieved nutrient values against the matched structured regulation rules.

For a retrieved regulation rule r, the calculator reads the normalized nutrient value v, the threshold value t_r, and the comparison operator op_r from the structured rule. Before the comparison, the value is aligned with the unit and basis specified in the structured rule when this information is available. The eligibility decision is computed as:

$$\text{eligible}(v, r) = \begin{cases} 1, & v \; op_r \; t_r, \\ 0, & \text{otherwise.} \end{cases}$$

The comparison is implemented in code rather than generated by the LLM.

This separation is important for enterprise use. It improves controllability, makes the reasoning path more inspectable, and reduces the risk of unsupported compliance conclusions.

4.7.4 Execution of orchestrated plans

For orchestrated dashboard queries, the framework executes the generated plan step by step. Each step is bound to a source, an operation, and a dependency rela-

tion. Source-specific execution modules are used to run each step, and intermediate results are recorded together with execution metadata.

After all steps are executed, the framework aggregates the outputs according to the specified merge strategy. Depending on the analysis type, this may involve left joins, sequential comparisons, concatenation of independent aggregate results, or calculation-specific aggregation logic. For example, some cross-source queries require source joining, while others only require parallel aggregation of independent source-level results.

This execution design separates planning from execution. The orchestration layer decides what should be done, while the execution layer carries out the plan over actual sources. This makes the framework more modular and easier to debug.

4.7.5 Answer synthesis

After the execution stage is complete, the framework generates the final answer. Answer synthesis is performed over grounded execution outputs and retrieved evidence rather than over the raw user query alone.

The answer generator receives structured execution results, regulation matches, compliance outputs, and the parsed intent. It then organizes these grounded results into a natural language answer. For simple questions, the answer may be a short value-focused response. For more complex queries, the answer may summarize multi-step execution outputs, cross-source comparisons, or identified drivers.

The framework also keeps step-level trace information throughout the pipeline. Each major step records action type, execution duration, source information, and intermediate outputs. This trace information supports debugging, system inspection, and later auditing. As a result, the final answer is not only useful to the end user, but also traceable for development and enterprise use.

5 Resources and datasets

In this chapter, we describe the main resources and datasets used in this thesis. We first introduce the system resources and their schema organization, including Power BI dashboards, open-source CSV data, and the structured regulation resource. We then describe how we construct the structured regulation resource used for regulation retrieval and compliance checking. After that, we introduce the public benchmark dataset CLINC150 [24] and explain how we adapt it to the schema-grounded setting used in this thesis. Finally, we present the enterprise 120-QA dataset and its annotation structure.

The goal of this chapter is not to explain the method itself. Instead, we explain what resources we use, what datasets we evaluate on, how these datasets are constructed, and how they are organized for later experiments.

Table 2: Main system resources used in this thesis.

Source Type	Source Name	Main Role
Power BI dashboard	`usnc_pns`	Nutrition compliance and Positive Nutrition Standards analysis, including overview and breakdown by country, brand, category, and time period
Power BI dashboard	`hfss_6npm`	UK HFSS profiling analysis, including score and HFSS / Non-HFSS status
Power BI dashboard	`ns_6npm`	Nutri-Score analysis, including grade and score distributions for product and portfolio analysis
Power BI dashboard	`terrepack`	Packaging sustainability analysis, including PAM weight sold, recyclability, PCR content, and material breakdown
CSV source	`zinc_deficiency`	Public zinc deficiency analysis based on WHO VMNIS data
CSV source	`iodine_deficiency`	Public iodine status analysis based on WHO VMNIS data
Regulation resource	`regulations.json`	Structured nutrition claims regulation resource for retrieval and compliance checking

5.1 System resources and source schemas

The system operates over a multi-source environment. In this thesis, three main types of resources are used: Power BI dashboards, open-source CSV data, and a structured regulation resource.

The primary execution backend consists of internal Power BI dashboards. The system queries Power BI semantic models in real time through API-based execution and normalizes the returned tables into a unified record format. In this thesis, four main dashboards are connected: `usnc_pns`, `hfss_6npm`, `ns_6npm`, and `terrepack`. To complement the internal dashboards, two open-source CSV data sources, `zinc_deficiency` and `iodine_deficiency`, are also integrated from the WHO Vitamin and Mineral Nutrition Information System (VMNIS) data portal.[5] In addition, a structured regulation resource stored as `regulations.json` is used to support regulation retrieval and compliance checking. Table 2 summarizes these resources and their main roles.

These resources differ not only in content, but also in structure and execution interface. The Power BI dashboards are queried through DAX over dashboard-specific semantic models. The CSV sources support filtering and aggregation over local tabular data. The regulation resource supports structured retrieval and rule matching rather than dashboard-style metric querying. Because of these differences, a single flat schema is not sufficient. The system must support both high-level source routing and source-level executable query construction. For this reason, a dual-layer schema design is used.

The first layer is a global data source registry. This layer describes what sources are available in the system and what each source can support. Each registry entry contains fields such as source identity, source type, description, routing cues, supported operations, schema linkage, and join information. These fields are included because the system needs more than a simple list of source names. It must identify candidate sources, match queries to source capabilities, and prepare for cross-source execution when necessary. Table 3 summarizes the main field groups in this layer.

[5] `https://www.who.int/teams/nutrition-and-food-safety/databases/vitamin-and-mineral-nutrition-information-system/data`

Table 3: Main field groups in the global data source registry.

Field Group	Main Role
Identity and type	Uniquely identify each source and distinguish source types
Description and routing cues	Support semantic source matching and routing
Capability and operation support	Indicate which analytical operations a source can support
Schema and join linkage	Connect each source to its local schema and support cross-source linking

Table 4: Main components in a source-specific schema.

Component	Main Role
`time`	Define valid time fields, formats, and default periods
`dimensions` and `attributes`	Define filterable and groupable source attributes with canonical field mappings
`slots`	Define the slot interface used by intent parsing and slot normalization
`measures` and `metrics`	Define executable quantitative targets for query generation
`default_output_fields`	Define default return fields for common operations such as list queries and detail lookup
`guardrails`	Restrict invalid field use and unsafe query generation patterns

This first layer functions as a routing-oriented schema. It is used to define the searchable source space before source-level execution begins.

The second layer is the source-specific schema. This layer is not designed only around raw source structure. Instead, it is designed around the operation space supported by the system, so that different query operations can be mapped to the corresponding slots, measures, output fields, and execution constraints. In practice, each source-specific schema contains the core components needed for executable querying. Table 4 summarizes these components.

The reason for this design is that different operations rely on different parts of the schema. For `query_value`, the main requirements are executable measures, filter slots, and time fields. For `list_item`, item identifiers and default output fields become more important. For `rank_by_dimension`, the schema must expose rankable dimensions and sortable measures. For `compare_values` and `trend_over_time`, grouping dimensions, period definitions, and comparable metrics are needed. For `get_product_detail`, the system depends on identifiers such as `cuc_code` or `sku_code` together with detail-oriented output fields. The source-specific schema therefore defines not only the valid field space of a source, but also the executable action space that can be applied to that source.

At runtime, these schemas support multiple modules at the same time. The intent recognizer uses them to constrain possible slots, supported operations, and candidate sources. The slot normalizer uses them to map extracted values to canonical fields and validate them against the allowed value space. The query composer uses them to locate executable measures, determine grouping fields, select default output fields, and apply source-specific constraints. Later execution modules rely on the same schema definitions to keep source access controlled and consistent. In this sense, the schema is not only descriptive metadata. It serves as a core interface between natural language understanding and executable analytics.

A simplified example is the schema of `ns_6npm`. This is an internal Power BI dashboard for Nutri-Score analysis of Unilever products. Nutri-Score is a front-of-pack

nutrition labelling scheme used to summarize the nutritional quality of products. The dashboard contains product- and portfolio-level information related to Nutri-Score grades, score distributions, and score components. In this schema, the time section defines the canonical report period field, the allowed period values, and the corresponding validation rules. The dimensions section defines attributes such as `market`, `region`, `brand`, `category`, `cuc_code`, and `sku_code`, each with a canonical field mapping, field description, and validation policy. For many attributes, the schema also defines the expected value space through enum values, examples, or validation rules. For example, `market` may include country values such as `Netherlands`, `France`, and `Germany`, while `region` may include values such as `Europe` and `North America`. These value definitions help the system identify slot values in natural language, normalize them to the canonical source representation, and use them as valid filter values in the generated DAX query. The schema also defines Nutri-Score-related attributes such as `ns_score`, `ns_total`, and the component fields `ns_energy`, `ns_safa`, `ns_sugar`, `ns_sodium`, `ns_fvn`, `ns_fibre`, and `ns_protein`. To support execution, it further specifies measures such as `ns_a_servings` to `ns_e_servings`, slot mappings for runtime parsing, default output fields for detail queries, and guardrails that constrain invalid DAX generation. This example shows that the schema is used as an executable representation of the source rather than as a passive field list.

The same design principle also applies to the other source types, although the internal details differ. CSV schemas define the filterable fields, normalization rules, and aggregation targets needed for structured tabular querying. The regulation resource defines structured retrieval and rule-matching fields such as region, claim name, product category, threshold text, and structured rule objects. Across all source types, the goal remains the same: to expose a controlled and executable interface between user intent and downstream source-specific operations.

5.2 Construction of the structured regulation resource

To support regulation retrieval and compliance checking, we construct a structured regulation resource and export it as `regulations.json`. We use this resource as the regulation backend of the system.

The original regulation materials come from two sources. The first source is existing internal tabular files. These files already contain basic regulatory information, such as region, product group, claim type, and threshold conditions. The second source is a set of original regulation PDF documents used to support and verify the structured entries.

We do not rebuild the regulation resource from scratch. Instead, we extend the existing internal tabular files and standardize them into a structured format. This format makes each entry executable in the sense that the system can directly use explicit fields, such as region, claim type, nutrient, and threshold conditions, for candidate filtering, rule matching and calculation. In particular, we add fields such as `reference` and `original text`. The `original text` field is first extracted from PDF documents with the help of an LLM, and we then manually review the

Table 5: Main fields in each structured regulation entry.

Field	Description
id	Unique identifier of the regulation entry
region	Regulation region, such as US, EU, or Codex
claim_name	Standard claim name
variants	Alternative wording variants of the same claim
food_category	Product category or basis condition, such as solids or liquids
regulation_text	Main regulation text used for interpretation
threshold_text	Human-readable threshold expression
reference	Source reference of the rule
full_citation	Full citation text for traceable display
structured_rules	Structured rule object containing nutrient, operator, threshold value, unit, per-amount basis, and condition type

extracted text. After that, we standardize the existing claim rules across regions and convert them into a unified JSON structure through scripts.

The final `regulations.json` file contains 467 structured regulation entries and covers three regulatory systems: US, EU, and Codex Alimentarius (Codex), which is used as an international food standards framework outside the US and EU settings. Each entry stores both text-level and rule-level information. At the text level, each record includes fields such as `id`, `region`, `claim_name`, `variants`, `food_category`, `regulation_text`, `threshold_text`, `reference`, and `full_citation`. At the rule level, each record contains a `structured_rules` object with fields such as `nutrient`, `operator`, `threshold_value`, `unit`, `per_amount`, and `condition_type`. Table 5 summarizes the main fields of the structured resource.

The main legal sources used in this resource are the relevant FDA claim standards for the US from 21 CFR Ch. I,[6] Regulation (EC) No 1924/2006 for the EU,[7] and Codex CAC/GL 23-1997.[8]

This resource serves two connected roles in the system. First, it supports regulation retrieval. The system uses fields such as region, claim type, nutrient, and product group to recall and rank candidate rules. Second, it supports compliance calculation. After the correct rule entry is retrieved, the structured threshold fields are used to compare user-provided or dashboard-retrieved nutrient values against the regulatory requirement. Therefore, `regulations.json` is not only a knowledge resource, but also a direct input to executable compliance checking.

5.3 Public benchmark dataset: CLINC150

In addition to the enterprise dataset, we use the public benchmark dataset CLINC150 [24]. We choose CLINC150 as the benchmark dataset for this thesis for three reasons. First, it is a widely used benchmark for intent recognition, which directly matches the core intent parsing problem studied in this work. Second, it contains 150 intent labels distributed across 10 domains, so the label space is large enough to provide a meaningful testbed for schema grounding and few-shot retrieval.

[6] https://www.govinfo.gov/content/pkg/CFR-2015-title21-vol2/html/CFR-2015-title21-vol2.htm

[7] https://eur-lex.europa.eu/eli/reg/2006/1924/oj/eng

[8] https://www.fao.org/4/y2770e/y2770e07.htm

Table 6: Official CLINC150 in-scope splits and their roles in this thesis.

Split	Official Size	Per Intent	Role in This Thesis
Train	15,000	100	Few-shot candidate pool and retrieval index
Validation	3,000	20	Not used in this thesis
Test	4,500	30	Source of the final evaluation subset
Evaluation subset	750	5	Final benchmark evaluation set, sampled from the official test split

Third, before moving to the enterprise setting, we use this public benchmark to validate whether the main method shows the same performance trend in a standard and controlled environment. In this way, the benchmark experiment can support the later enterprise experiments rather than being treated as a separate task.

CLINC150 contains 150 intent labels distributed across 10 domains. The official release provides train, validation, and test splits. Each intent has 100 training samples, 20 validation samples, and 30 test samples. Table 6 summarizes the official split sizes and their roles in this thesis.

The original CLINC150 data only provides query-label pairs. Each record contains the user query text and the intent label, but it does not provide any intent description, keyword hints, or schema-like semantic information. This is different from our enterprise setting, where the parser uses explicit schema context during prompting. If we directly gave the model only the 150 raw intent labels, the setting would no longer match the schema-grounded design studied in this thesis.

This would create two practical difficulties. First, some labels are not fully transparent from the label string alone, especially abbreviation-like labels such as `apr` or `w2`. Second, some intents are semantically close and therefore harder to distinguish from label names alone, such as `credit_score` versus `credit_limit`, or `bill_due` versus `pay_bill`. Similar ambiguity can also appear in the kitchen and dining domain, where a query about `nutrition_info` must still be distinguished from other food-related requests. In this case, the benchmark setting would be closer to raw label-list classification than to the schema-grounded prompting setting that we want to study.

For this reason, we construct an auxiliary schema for CLINC150. The goal is not to change the original benchmark labels, but to make the dataset compatible with schema-grounded prompting. For each of the 150 intent labels, we create a short natural language description and four representative keywords. We keep the original CLINC150 labels unchanged and only add auxiliary semantic information around them. In this way, the public benchmark remains methodologically aligned with the schema-grounded design later used in the enterprise setting.

We construct this auxiliary schema through batch generation with assistance from GPT-5.4 in a web-based ChatGPT interface[9] followed by manual review. Specifically, we extract the training data from the official `train` split in JSON format and use it together with the intent labels and domain metadata as input to the model. For each intent, we then ask the model to generate one short description

[9] `https://chatgpt.com/`

Table 7: Structure of each CLINC150 schema entry in `e1_schema.json`.

Field	Description
`label`	Original CLINC150 intent label
`domain`	Domain to which the intent belongs
`description`	Short natural language description of the user intent
`keywords`	3 - 4 representative trigger phrases or expressions for this intent

and three to four representative keywords as draft schema entries. We manually review the generated entries with reference to the corresponding training utterances to ensure that the description reflects the main intent meaning and that the keywords are representative and sufficiently discriminative. The final schema is stored as `e1_schema.json`. Each schema entry contains four fields: the original label, the domain, a short description, and three to four representative keywords. Table 7 summarizes this structure.

A simplified example is shown below:

`label`: `credit_score`
`domain`: `banking`
`description`: Ask about personal credit score or rating information.
`keywords`: credit score, credit rating, credit report, FICO score

In our study, the official train split is not used for model training or fine-tuning. Instead, we use it for two purposes. First, it supports the construction of the auxiliary schema described above. Second, it serves as a few-shot resource: representative examples can be selected from the train split for static few-shot prompting, and the same split can also be encoded into an embedding index for dynamic retrieval.

We do not use the full 4,500 test queries for final evaluation. Instead, we construct a balanced evaluation subset through stratified sampling. We group the official test split by intent label, sample 5 queries from each intent with a fixed random seed of 42, and merge the sampled queries into one evaluation set. This gives $150 \times 5 = 750$ evaluation queries. We use stratified sampling instead of simple random sampling to preserve equal label coverage across all intents, which is important for macro-level evaluation. We also use a fixed seed to make the benchmark setting reproducible. The final 750-query subset sampled from the official test split is used for formal evaluation.

Therefore, CLINC150 serves as a public benchmark for validating the intent parsing method outside the enterprise dataset. After we augment it with an auxiliary schema, it also becomes a compatible testbed for studying schema-grounded prompting and dynamic few-shot retrieval in a controlled public setting.

5.4 Construction of the enterprise 120-QA dataset

To evaluate structured intent parsing and executable question answering in a realistic enterprise setting, we construct an enterprise 120-QA dataset. The dataset is designed to reflect actual nutrition and health analysis needs in the business context studied in this thesis.

We first collect 20 seed questions through a questionnaire-based process with internal teams. For each seed item, we collect not only the question itself, but also the relevant data source, the main workflow steps, and the expected answer. These seed questions represent common business tasks, such as dashboard metric lookup, product-level analysis, regulation lookup, compliance checking, and multi-source analytical workflows.

We then manually expand these seed questions into the final 120-question dataset in two main ways. First, we vary the surface form of the questions by creating paraphrased, rewritten, and typo-style variants. Second, we vary the structural conditions of the questions to ensure coverage of different question categories, dashboards, abstract operations, and execution modes. This expansion is guided by the characteristics of the connected data sources. In particular, we create additional questions by instantiating similar analytical intents over different source-specific metrics, dimensions, and filter combinations, so that the dataset covers the main executable patterns supported by each source. For example, a ranking question over countries in one dashboard can be expanded into ranking questions over brands, regions, or other source-specific dimensions in another dashboard, under different filter settings. We then ask nutrition scientists to review whether the resulting questions are reasonable and consistent with real business questioning patterns. After that, we manually verify whether the answer to each question can be found in the corresponding dashboard, so that each sample forms a complete QA pair before the annotation step.

We then annotate the questions with structured labels. After annotation, the dataset is reviewed again by a nutrition scientist for consistency. The final dataset is an annotated table that contains the question text, verified answer information, and structured labels for later module-level and end-to-end evaluation. We then import this annotated table into Python for further processing, including data cleaning, split generation, and experiment preparation.

The final dataset contains 120 question samples. We split it into 31 development samples and 89 validation samples using a group-aware strategy. Variants derived from the same original seed question do not appear across both splits. In this way, we avoid leakage caused by highly similar paraphrases or typo variants. The development split is mainly used for prompt development and retrieval-pool construction, while the validation split is used for formal evaluation.

Table 8 summarizes the dataset distribution across split, question category, abstract operation, query complexity, and variant.

The dataset is dominated by `data_query` questions, which account for roughly two thirds of the samples, while `compliance_check` forms the second largest cate-

Table 8: Distribution of the enterprise 120-QA dataset.

Category	**Attribute**	**Total**	**Dev**	**Val**
Split	–	120	31	89
Question category	`data_query`	79	18	61
	`compliance_check`	30	9	21
	`regulation_lookup`	7	3	4
	`meta`	4	1	3
Abstract operation	`check_eligibility`	30	9	21
	`get_product_detail`	18	4	14
	`query_value`	16	4	12
	`compare_values`	16	4	12
	`rank_by_dimension`	13	2	11
	`list_item`	10	3	7
	`get_regulation_standard`	7	3	4
	`trend_over_time`	6	1	5
	`other`	4	1	3
Query complexity	`simple`	108	27	81
	`complex`	12	4	8
Variant	`orig`	107	30	77
	`para`	8	1	7
	`typo`	5	0	5

gory. `regulation_lookup` and `meta` are smaller, but they are still needed to cover the full task space of the system. The operation distribution shows that all nine predefined abstract operations are covered. Among them, `check_eligibility` is the most frequent, followed by `get_product_detail`, `query_value`, and `compare_values`. In terms of complexity, most queries are `simple`, while `complex` queries account for only a small portion. This reflects the actual business setting, where most questions are single-source and single-step, but a smaller number still require more advanced workflows.

The dataset also covers the main source types of the system. Most questions are related to `usnc_pns` and regulation-based workflows, while smaller subsets involve `terrepack`, `hfss_6npm`, `ns_6npm`, and the two CSV sources. Since some questions require more than one source, especially in cross-source settings, source counts are not used here as the main summary statistic. Instead, the table focuses on the label dimensions that are most directly relevant to the later evaluation.

Overall, this dataset is not a general open-domain QA resource. It is a task-oriented enterprise dataset constructed specifically for structured intent parsing, regulation retrieval, and executable multi-source question answering. The full question set is not included in the appendix because some questions and slot fields are

based on internal product-related data and non-public market information.

5.5 Annotation structure and evaluation targets

Each sample in the enterprise 120-QA dataset is associated with a structured gold annotation. The annotation table serves as a shared resource for several downstream modules and later evaluations.

The most important layer of this table is the structured intent layer, which consists of `question_category`, `abstract_operation`, `data_sources`, and `slots`. Together, these fields define the main structured interpretation of the query. The table also retains `query_complexity` and `variant`, which are mainly used for subgroup analysis and robustness evaluation rather than as primary prediction targets. In the annotation table, `query_complexity` is equivalent to the execution-mode distinction introduced earlier, but is expressed here as a dataset label.

Among these fields, `slots` is the central structured field. The original annotation table retains both a unified `slots_json` representation and a set of decomposed slot columns, such as `period`, `market`, `region`, `brand`, `product_group`, `cuc_code`, `claim_type`, `nutrient`, and `regulation_region`. In this thesis, we treat `slots_json` as the canonical slot representation. Each key represents a slot type, that is, a *slot key*, and each value represents the corresponding normalized slot value. The decomposed slot columns are mainly retained for analysis and later evaluation support.

We annotate slots because later execution depends on explicit executable parameters rather than on general semantic similarity alone. For dashboard-oriented questions, these parameters usually correspond to filters, grouping conditions, or product identifiers. For regulation-related questions, they correspond to structured constraints needed for regulation retrieval and threshold checking. We annotate only the parameters that are explicitly expressed in the query and needed for later execution, and we do not add unsupported values that are not grounded in the query text.

For later structured intent evaluation, `slots` is not treated as one undifferentiated field. Instead, it is assessed at the level of slot keys and slot values. In particular, we focus on a predefined set of execution-critical slot types, namely `period`, `market`, `region`, `brand`, `product_group`, `cuc_code`, `claim_type`, `nutrient`, and `regulation_region`. These slot types are treated as key slots in the later slot-related evaluation, with `slots_json` serving as the canonical gold reference.

The slot structure varies across question categories. For `data_query` samples, slots mainly correspond to dashboard filters, grouping fields, or product identifiers. For `compliance_check` samples, the slot structure is richer and often includes regulation-related constraints together with a nested `nutrient_values` array. For `regulation_lookup` samples, the slot structure is similar to `compliance_check`, but it usually does not include `nutrient_values`, because the task is to retrieve the rule itself rather than compare a product value against that rule.

In addition to the core structured intent fields, the annotation table also con-

tains supporting fields for different downstream modules. Some samples have a `golden_regulation_id`, which serves as the gold target for regulation retrieval. Some samples have an `answer`, which serves as the gold reference for final answer verification. The table also includes execution-oriented supporting columns, such as regulation, calculation, and orchestration flags. In addition, it includes output-control information such as metrics, output fields, ranking dimensions. These columns are not all direct prediction targets in later experiments, but they help connect natural language queries to later execution logic.

Overall, the enterprise annotation table functions as a unified annotated resource shared across modules. The structured intent parsing module mainly uses the core intent fields. The regulation retrieval module mainly uses regulation-related samples together with `golden_regulation_id`. The final answer verification module mainly uses `answer` and the normalized gold-answer resource derived from it. In this sense, different modules use different columns and, when needed, different row subsets of the same annotation table.

6 Experimental setup

6.1 Overview of the experimental design

The proposed system contains multiple modules, but not every module is evaluated in isolation. To keep the evaluation focused and aligned with the research questions, we organize the experiments into four connected parts.

The first experiment uses CLINC150 as a public benchmark. Its role is to validate the core method in a standard and controlled setting before moving to the enterprise environment.

The second and third experiments evaluate two intermediate capabilities in the enterprise setting that are most critical for later system performance. The second is structured intent parsing, which determines whether the system can correctly identify the task type, the required operation, the relevant data sources, and the key execution parameters. The third is regulation retrieval, which determines whether the system can locate the correct regulation entry before compliance reasoning is performed.

The fourth experiment evaluates the complete end-to-end executable question answering pipeline. It tests whether improvements in the intermediate modules translate into correct and traceable final outputs in realistic business scenarios.

Together, these four experiments form a progressive evaluation design: public-benchmark validation, enterprise-level intermediate evaluation, and end-to-end system evaluation. This design allows us to assess not only whether the proposed method works, but also how improvements in key intermediate stages contribute to overall system performance.

6.2 Experiment 1: Public benchmark validation on CLINC150

6.2.1 Objectives

This experiment evaluates the core intent parsing method on a public benchmark in a standard and controlled setting. It focuses on three goals:

- whether schema information helps the model distinguish among a large set of intent labels;
- whether dynamically retrieved examples are more useful than no examples or a fixed example set;
- whether the retrieval-based prompting strategy also works on a public benchmark outside the enterprise setting.

6.2.2 Data

Section 5.3 has already described the CLINC150 benchmark, the auxiliary schema, and the roles of the train and test splits in this experiment, so we do not repeat these details here.

6.2.3 Compared methods

We compare four methods in this experiment. The comparison isolates two factors: whether schema information is injected into the prompt and how few-shot examples are constructed.

The first method is **ICL no-schema**. This is a retrieval-based in-context learning baseline adapted from prior work on retrieval-augmented intent classification. It uses dynamically retrieved examples but does not inject schema information [13]. We include it as a conceptually relevant published baseline. We do not claim exact reproduction of the original reported numbers. Instead, we adapt the same general retrieval-based no-schema idea to our CLINC150 subset for direct comparison with the proposed schema-grounded method.

The second method is **B1**, a schema-grounded zero-shot baseline. It injects the full intent schema into the prompt but does not use any examples. This baseline isolates the effect of schema information alone.

The third method is **B2**, a schema-grounded static few-shot baseline. It uses the same full schema as B1, but augments it with a fixed set of training examples selected in advance. The same example set is reused for all test queries. To keep the comparison aligned with the dynamic retrieval setting, we construct fixed example sets for $k = 1$, $k = 3$, and $k = 5$ by first randomly selecting k distinct domains with random seed 42 and then sampling one training example from each selected domain. Because the seed is fixed, the resulting example set under each k is random but fully reproducible.

The fourth method is **P**, the proposed method. It combines full schema grounding with query-specific dynamic retrieval. For each test query, it retrieves the top-k most

Table 9: Comparison of the four methods in Experiment 1 on CLINC150.

Method	Use of training split	Schema context	Few-shot strategy	Retrieval input	Retrieved examples	Prompt components
ICL no-schema	The training queries are used to build a retrieval index. Gold labels are used only to format retrieved examples.	None	Dynamic few-shot	Training queries and current evaluation query	Top-k semantically nearest training examples represented as query–label pairs	Instruction, retrieved few-shot examples, and input query
B1	The training split is not used for retrieval or example construction.	Full schema	None	None	None	Instruction, full 150-intent schema, and input query
B2	The training split is used to construct a fixed few-shot set in advance. The same examples are reused for all test queries.	Full schema	Static few-shot	None	None	Instruction, full schema, fixed few-shot examples, and input query
P	The training queries are used to build a retrieval index. Gold labels are used only to format retrieved examples.	Full schema	Dynamic few-shot	Training queries and current evaluation query	Top-k semantically nearest training examples represented as query–label pairs	Instruction, full schema, retrieved few-shot examples, and input query

similar training examples and injects them into the prompt together with the full schema.

Table 9 summarizes the differences among the four methods in terms of training-split usage, schema context, few-shot strategy, retrieval behavior, and prompt composition.

All four methods solve the same final task and produce the same output format, namely one intent label. Their main differences lie in whether schema information is provided and whether the examples are absent, fixed, or dynamically retrieved.

6.2.4 Evaluation metrics

We use two standard classification metrics: Accuracy and Macro-F1. Accuracy is the fraction of test queries whose predicted label exactly matches the gold label:

$$\text{Accuracy} = \frac{N_{\text{correct}}}{N}.$$

For each intent label c, we compute its F1 score as:

$$F1_c = \frac{2 \cdot P_c \cdot R_c}{P_c + R_c},$$

where P_c and R_c are the precision and recall for that label. Macro-F1 is then computed as the unweighted average of the class-level F1 scores across all 150 intent labels:

Table 10: Main experimental settings for Experiment 1.

Setting	Specification
Base classifier	`gpt-3.5-turbo` via the OpenAI API
Embedding model	`text-embedding-3-small`
Output format	One intent label
Temperature	0.1
Maximum output	20 tokens

$$\text{Macro-F1} = \frac{1}{150} \sum_{c=1}^{150} F1_c.$$

We report both metrics to reflect overall correctness and class-balanced performance.

6.2.5 Experimental protocol

Table 10 summarizes the main experimental settings used in Experiment 1. The OpenAI model and embedding model are accessed through the OpenAI platform.[10][11]

All methods use the same 750-query benchmark subset so that the comparison isolates prompt structure and example selection strategy. No fine-tuning or parameter update is performed. The system prompt requires the model to output exactly one intent label, and a lightweight normalization step is applied after generation when needed.

6.3 Experiment 2: Structured intent parsing on the enterprise dataset

6.3.1 Objectives

This experiment evaluates structured intent parsing on the enterprise dataset. The evaluation examines:

- whether dynamic few-shot retrieval improves structured intent parsing compared with no-retrieval and static few-shot prompting;
- whether the prompting strategies remain robust across query complexity and surface-form variation;
- comparing the token-efficiency trade-offs of the prompting settings.

[10] `https://platform.openai.com/docs/models`
[11] `https://platform.openai.com/docs/guides/embeddings`

6.3.2 Data

This experiment uses the enterprise 120-QA dataset introduced in Sections 5.4 and 5.5. The development split is used for prompt-side example construction, while the validation split is used for final evaluation. In this experiment, the main prediction targets are the core structured intent fields, namely `question_category`, `abstract_operation`, `data_sources`, and `slots`. We also use `query_complexity` and `variant` for subgroup analysis.

6.3.3 Compared methods

We compare three schema-grounded methods in this experiment: **B1**, **B2**, and **P**. All three methods share the same structured output format, the same schema-grounded prompt framework, the same registry-based dashboard filtering step, and the same post-processing pipeline. The only controlled difference is how few-shot examples are constructed and injected.

Before each LLM call, the system applies registry-based filtering to identify source-relevant dashboards through lightweight keyword matching over registry metadata. More specifically, the query is matched against the `signature_metrics` field of each dashboard in the global data source registry. Primary keyword matches receive a higher score than secondary matches, and only the top-scoring dashboards are retained. The detected dashboards determine which source-specific schemas are injected into the prompt. This step narrows the schema context and reduces token usage, but it does not directly determine the final routing result. The final routing prediction is produced by the LLM as part of the structured intent output. Since this filtering step is shared by B1, B2, and P, the comparison isolates the effect of few-shot strategy rather than source scoping itself.

The first method is **B1**, a schema-grounded no-retrieval baseline. It uses the shared prompt framework and source-relevant schema context, but it does not inject any few-shot examples.

The second method is **B2**, a schema-grounded static few-shot baseline. It uses the same schema context as B1, but augments the prompt with a fixed set of examples selected in advance from the development split. We evaluate $k \in \{1, 3, 5\}$. The fixed example sets are manually chosen to provide broader slot coverage and category diversity. The selected example IDs are $[8]$ for $k = 1$, $[8, 77, 69]$ for $k = 3$, and $[8, 62, 31, 69, 77]$ for $k = 5$. These examples are reused for all validation queries under the corresponding k. Each example is injected in the same compact structured format used by the dynamic retrieval setting.

The third method is **P**, the proposed method. It uses the same schema-grounded prompt framework, but retrieves query-specific examples from the development split at inference time. For dynamic retrieval, we build a fixed embedding index over the query text of the 30 original-form development samples only. The structured output JSON is not used in similarity computation. This design keeps the retrieval pool canonical, while the full validation split is used to evaluate both overall performance

Table 11: Comparison of the three methods in Experiment 2.

Method	Schema context	Few-shot strategy	Example pool	Retrieval basis	Main prompt difference
B1	Scoped schema	None	None	None	Prompt contains only the shared task instructions and source-relevant schema context
B2	Scoped schema	Static few-shot	Fixed examples from the Dev split	None	Prompt contains the shared schema context plus a fixed compact example set for the chosen k
P	Scoped schema	Dynamic few-shot	30 original-form Dev samples	Query-text embedding retrieval with cosine similarity	Prompt contains the shared schema context plus top-k query-specific compact examples retrieved from the indexed Dev pool

and robustness. For each validation query, we embed the current query, compute cosine similarity against all indexed development queries, and retrieve the top-k nearest samples. The retrieved samples are then injected into the prompt together with the scoped schema context in the same compact structured format used by B2. We evaluate $k \in \{1, 3, 5\}$.

Unlike Experiment 1, which injects the full intent schema for all labels, this experiment uses scoped schema injection. Only the source-relevant schema context identified by registry-based filtering is loaded into the prompt. This design better matches the enterprise setting, where the number of possible fields, slots, and measures is much larger than in the public benchmark.

Table 11 summarizes the differences among the three methods.

6.3.4 Evaluation metrics

We evaluate structured intent parsing at both the field level and the combined intent level. The main metrics cover category prediction, operation prediction, routing quality, slot extraction quality, and overall structured intent correctness. In addition, we report average prompt tokens per query and a token-efficiency score to compare the efficiency of different prompting strategies.

Table 12 summarizes the main metrics used in this experiment.

Among these metrics, Core EM is the main combined metric. For each query, Core EM is assigned a value of 1 only when `question_category`, `abstract_operation`, `data_sources`, and all gold key slots are correct after normalization and post-processing; otherwise, it is assigned 0. The reported Core EM is the average over all evaluation queries. This makes it stricter than field-level accuracy and more informative for later routing and execution.

Token Efficiency is computed as:

$$\text{Token Efficiency} = \frac{\Delta\text{Core EM}}{\Delta\text{Prompt Tokens}/1000},$$

Table 12: Main evaluation metrics for Experiment 2.

Metric	Target / Scope	Definition
Category Accuracy	`question_category`	Exact-match accuracy of the predicted question category
Operation Accuracy	`abstract_operation`	Exact-match accuracy of the predicted abstract operation
Routing F1	`data_sources`	Set-based F1 over the predicted data-source set
Routing Exact Match	`data_sources`	Exact set match of the predicted data-source set
Slot F1	Gold key slots in `slots_json`	F1 over the predicted slot-key set against the gold key-slot set
Slot Value Acc	Values of gold key slots	Average exact-match accuracy of the values of the 9 execution-critical key slots
Core EM	Core intent fields + gold key slots	Exact match of `question_category`, `abstract_operation`, `data_sources`, and all gold key slots present in `slots_json`
Avg Token	Per query	Mean number of prompt tokens per query
Token Efficiency	Relative to B1	Gain in Core EM divided by the additional prompt tokens relative to the B1 baseline

Table 13: Main experimental settings for Experiment 2.

Setting	Specification
Base classifier	Azure OpenAI GPT-4o
Embedding model	`text-embedding-3-small`
Output format	Structured JSON object
Temperature	0.1
Maximum output	4096 tokens

where ΔCore EM and ΔPrompt Tokens are measured relative to the B1 baseline.

6.3.5 Experimental protocol

Table 13 summarizes the main experimental settings used in Experiment 2. The base model is accessed through the Azure OpenAI deployment available in the enterprise environment. The same OpenAI embedding model as in Experiment 1 is used for dynamic example retrieval.

All methods use the same 89-query validation split, the same base model, the same source-scoping mechanism, and the same post-processing pipeline. This controlled setup ensures that the comparison isolates the effect of few-shot strategy. B1 provides the no-retrieval baseline, B2 tests whether a small but fixed set of examples is sufficient, and P tests whether query-specific example retrieval yields better structured intent parsing under the same overall framework.

For dynamic retrieval, we build an embedding index over the 30 original-form development queries and retrieve the top-k nearest neighbors for each validation query. The retrieved examples are then injected into the shared prompt in a compact structured format. For B2, the fixed examples are also injected in the same compact format so that the comparison changes the example source, not the example representation.

No parameter update or fine-tuning is performed in this experiment. All predic-

tions are evaluated after the shared post-processing stage, because the system uses post-processed intent outputs rather than raw LLM strings in later execution.

6.4 Experiment 3: Regulation retrieval evaluation

6.4.1 Objectives

This experiment evaluates the regulation retrieval module. Given a natural-language nutrition-claim query, the module should retrieve the correct regulation entry or entries from the structured regulation resource. The evaluation has two main goals:

- to compare different retrieval strategies for finding the correct regulation entries;
- to test whether the proposed hybrid retrieval method improves Top-1 rule selection.

6.4.2 Data

This experiment uses the regulation-related subset of the enterprise annotation table introduced in Chapter 5. Specifically, we select samples with IDs 68–104 and retain only the original-form variants, which results in 35 evaluation queries. Each sample contains a natural-language query and one or more gold regulation IDs. The structured constraints required for regulation retrieval are provided in the annotation table as manually annotated gold slots. In this experiment, these gold slots are given directly to the regulation retriever rather than predicted by the upstream parsing module. The resulting evaluation therefore reflects the upper-bound performance of the retrieval module under ideal slot conditions. This subset is used only for evaluation and is not used for rule refinement or model tuning.

6.4.3 Compared methods

We compare four retrieval methods: **B1**, **B2**, **B3**, and **P**. The comparison is designed to separate pure lexical retrieval, rule-based slot-aware retrieval, lexical-semantic hybrid retrieval, and the proposed hybrid design that combines rule-based recall with semantic reranking.

The first method is **B1**, a BM25 lexical retrieval baseline based on probabilistic lexical matching [21]. We build a BM25 index over all 467 regulation entries in the regulation knowledge base. For each regulation entry, we concatenate key textual fields, including claim name, wording variants, nutrient, region, food category, threshold or standard text, and source citation. We then tokenize the concatenated text and use BM25 to rank entries for each input query. B1 uses only the raw query text and does not use any structured slots.

The second method is **B2**, a rule-based slot-aware retrieval baseline. This method uses the existing regulation retriever together with manually annotated gold slots

Table 14: Comparison of the four methods in Experiment 3.

Method	Retrieval basis	Use of gold slots	Candidate generation	Ranking mechanism
B1	Query text only	No	Full regulation knowledge base	BM25 lexical ranking
B2	Query text + hard slot constraints	Yes	Rule-based filtering over the regulation knowledge base	Rule order after slot-aware filtering
B3	Query text only	No	Full regulation knowledge base	Hybrid BM25 + SBERT ranking
P	Query text + hard slot constraints	Yes	Rule-based recall with a broader candidate pool	Slot-aware SBERT cosine-similarity reranking

as hard constraints. It applies three main filtering steps: claim-type matching, regulation-region filtering, and product-group compatibility filtering. This design is highly interpretable and already provides a strong precision-oriented baseline. However, when several mutually compatible variants remain in the candidate set, the Top-1 ranking may still be unstable.

The third method is **B3**, a lexical-semantic hybrid baseline. Like B1, it retrieves from the full regulation knowledge base without using structured slots. It combines BM25 lexical ranking [21] with SBERT-based semantic similarity [22]. This method is included to test whether a simple lexical-semantic fusion can improve retrieval without using manually annotated slot constraints.

The fourth method is **P**, the proposed hybrid method. It also uses the manually annotated gold slots as structured input, but extends B2 with semantic reranking. The method operates in two stages. In the first stage, it uses the same rule-based retrieval logic as B2 to generate a regulation-compatible candidate set, but keeps a broader candidate pool in order to preserve recall. In the second stage, it applies slot-aware semantic reranking. We encode all 467 regulation entries in advance with the `all-MiniLM-L6-v2` Sentence-BERT model [22].[12] On the query side, we concatenate the raw query text with the gold slots using a slot-augmented representation and encode the result with the same model. We then compute cosine similarity between the query representation and the candidate regulation representations and rerank the candidate set accordingly. If the rule-recalled candidate set is still smaller than the requested cutoff, we supplement it with additional dense candidates from the full regulation pool.

Table 14 summarizes the main differences among the four methods.

6.4.4 Evaluation metrics

We evaluate regulation retrieval with three standard retrieval metrics: Precision@k, Recall@k, and Hit@k. All metrics are macro-averaged over queries and are reported at $k \in \{1, 3, 5\}$.

[12] `https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2`

Table 15: Evaluation metrics for Experiment 3.

Metric	**Scope**	**Definition**
Precision@k	Top-k retrieved entries	Proportion of the top-k retrieved regulation entries that are gold entries
Recall@k	Gold regulation set	Proportion of the gold regulation entries that are covered by the top-k retrieved results
Hit@k	Top-k retrieved entries	Whether at least one gold regulation entry appears in the top-k results

Table 15 summarizes the metrics used in this experiment.

For each query q, let G_q be the set of gold regulation entries and R_q^k be the top-k retrieved entries. The query-level metrics are computed as:

$$\text{Precision@}k(q) = \frac{|R_q^k \cap G_q|}{|R_q^k|},$$

$$\text{Recall@}k(q) = \frac{|R_q^k \cap G_q|}{|G_q|},$$

$$\text{Hit@}k(q) = \begin{cases} 1, & \text{if } |R_q^k \cap G_q| > 0, \\ 0, & \text{otherwise.} \end{cases}$$

The final reported score for each metric is the average of the query-level scores.

In this experiment, $k = 1$ reflects direct answer usability, where the system returns one regulation entry as its main answer. In contrast, $k = 3$ and $k = 5$ reflect candidate-set coverage for downstream reasoning and answer generation.

6.4.5 Experimental protocol

The regulation knowledge base contains 467 regulation entries covering three regulatory systems: EU, US FDA, and Codex nutrition-claim standards. The legal sources, detailed composition, and field structure of this resource are described in Section 5.2.

We evaluate retrieval performance at $k \in \{1, 3, 5\}$. These cutoffs reflect different usage scenarios. In particular, $k = 1$ corresponds to the setting in which the system returns a single regulation entry as its direct answer, whereas $k = 3$ and $k = 5$ correspond to settings in which the system provides a candidate set for downstream reasoning.

For input configuration, B2 and P use manually annotated gold slots as structured input. This design decouples the retrieval module from the upstream slot extraction module and allows us to evaluate the upper-bound performance of the retriever under ideal slot conditions, independently of Experiment 2. In contrast, B1 and B3 do not use any structured slots and rely only on the raw query text.

6.5 Experiment 4: End-to-end executable QA evaluation

6.5.1 Objectives

This experiment evaluates the complete end-to-end pipeline from natural-language query to final system answer. Unlike Experiment 2, which evaluates structured intent parsing in isolation, this experiment jointly evaluates intent parsing, source-specific execution, and answer generation under realistic enterprise nutrition and health compliance queries. The evaluation focuses on two aspects:

- the correctness of final answers produced by the full executable QA pipeline;
- the transfer of intent-recognition improvements to end-to-end system performance.

6.5.2 Data

This experiment uses the validation split of the enterprise 120-QA dataset introduced in Sections 5.4 and 5.5. The full validation split contains 89 samples. Three validation samples are multi-step analysis cases with IDs 106–108. We exclude these three cases from automatic evaluation because they do not have structured gold answers and are difficult to compare with one script-based matcher. Therefore, the final automatic evaluation set contains 86 samples, including 81 simple queries and 5 complex queries.

The automatically evaluated subset covers a range of operation types, including `query_value`, `rank_by_dimension`, `list_item`, `compare_values`, `trend_over_time`, `get_product_detail`, `get_regulation_standard`, `check_eligibility`, and out-of-scope cases. The development split is not used for evaluation in this experiment.

The end-to-end gold supervision is derived from the answer-bearing subset of the shared annotation table. In particular, we convert the original `answer` field into a normalized gold-answer resource, denoted as `golden_answer.json`, through a script-based transformation. This resource stores operation-specific structured target outputs that can be compared automatically against system predictions.

The gold-answer format is operation-dependent. Scalar queries are represented with fields such as query type, operation, target metric, and normalized value. Ranking queries additionally store the target dimension, ranking size, ordering mode, returned items, and unit. Other operations, such as regulation lookup and eligibility checking, use their own normalized answer structures. This design allows the end-to-end evaluation to compare final outputs under operation-specific rules rather than through one uniform string match.

In this setting, the downstream execution modules are kept fixed, including data-query execution, answer generation, and the regulation retrieval configuration. The experiment only varies the intent-recognition strategy and measures how this change affects final end-to-end system outputs.

Table 16: Compared settings in Experiment 4.

Setting	**Example context**	**Evaluated at**	**Main difference**
No Retrieval	None	one setting only	Zero-shot intent recognition without example context
Static Few-shot	Fixed examples	$k \in \{1, 3, 5\}$	Uses the same manually selected examples for all queries
Dynamic Retrieval	Query-specific examples	$k \in \{1, 3, 5\}$	Retrieves semantically similar examples from the development split for each query

6.5.3 Compared methods

This experiment compares different intent-recognition strategies while keeping the other major pipeline components fixed. The only variable is the prompting strategy used in the intent-recognition module.

The first setting is **No Retrieval**. It uses the same schema-grounded intent prompt as the other settings, but does not add any example context. It therefore corresponds to zero-shot structured intent prediction within the end-to-end pipeline.

The second setting is **Static Few-shot**. It uses a fixed set of manually selected examples as context. We evaluate this setting at $k \in \{1, 3, 5\}$.

The third setting is **Dynamic Retrieval**. It uses semantic retrieval over the development split to select query-specific examples and inject them into the prompt. We also evaluate this setting at $k \in \{1, 3, 5\}$.

To keep intent prompting as the only variable, the downstream execution modules are kept fixed in this experiment, including data-query execution, answer generation, compliance calculation, and regulation retrieval. The regulation retrieval component uses the hybrid retrieval configuration selected from Experiment 3. Therefore, the end-to-end comparison does not test different regulation retrieval variants again. Instead, it evaluates whether different intent-recognition strategies lead to different final system outputs under the same downstream execution setting.

Table 16 summarizes the compared settings.

6.5.4 Evaluation metrics

The main metric in this experiment is End-to-End Pass Rate (E2E Pass Rate), defined as the proportion of automatically evaluated samples for which the final system output matches the gold structured answer:

$$\text{E2E Pass Rate} = \frac{N_{\text{pass}}}{N_{\text{evaluated}}},$$

where N_{pass} is the number of samples that pass the operation-specific evaluation rule, and $N_{\text{evaluated}}$ is the total number of automatically evaluated samples.

Since the final answer format differs across operations, pass or fail is not determined by one uniform string-matching rule. Instead, each sample is judged according to the rule associated with its gold `operation`.

Table 17: Operation-specific judgement rules for Experiment 4.

Operation	Pass criterion
`query_value`	Normalized metric-name match plus numeric tolerance; percentage and decimal formats are treated as equivalent when applicable
`rank_by_dimension`	Strict mode requires correct order, keys, and values; relaxed mode compares the returned key–value set while ignoring order
`list_item`	Compare predicted and gold items as sets and ignore order
`compare_values`	Align returned items by normalized comparison keys and compare aligned values under the same numeric tolerance rule
`trend_over_time`	Align returned series by time period and compare them under a two-step matching strategy based on metric identity and value consistency
`get_product_detail`	Require exact match on mandatory key–value fields; list-valued fields are compared as unordered multisets
`get_regulation_standard`	Require gold regulation-ID coverage together with exact match of the final judgement string
`check_eligibility`	Support comparison of single decision, decision-by-region, and decision-by-claim formats, together with regulation-ID coverage checks
`other` / out-of-scope	Require correct rejection behaviour and the corresponding out-of-scope marker

Table 17 summarizes the operation-specific pass criteria used in automatic evaluation.

For numeric comparison, we use a two-level tolerance rule. For ratio-like values with absolute magnitude less than or equal to 1, we use an absolute tolerance. For larger numeric values, we use a relative tolerance. The evaluator also supports equivalent percentage and decimal formats when needed.

In addition to the overall pass rate, we record failure categories to identify where the pipeline breaks. Failure attribution follows a *first-failure* principle: once an earlier stage fails, later stages are not considered for error labeling. The attribution order is:

execution layer → intent layer → regulation layer → calculation layer → final answer layer.

This order makes the failure categories mutually exclusive and avoids assigning final-answer errors when an earlier execution or intent error has already made the answer invalid.

Table 18 summarizes the failure categories and their trigger conditions.

6.5.5 Experimental protocol

For each valid query in the evaluation set, we execute the end-to-end pipeline by sending the natural-language question to the deployed API together with the configuration of the intent method and, when applicable, the retrieval size k. The API returns a JSON response containing fields such as `answer`, `structured_result`, and `intent`. We then convert the response into a normalized structured answer with an operation-aware extractor and compare it against the gold-answer resource under the operation-specific rules described above.

The comparison covers seven configurations in total: one no-retrieval setting, three static few-shot settings ($k = 1, 3, 5$), and three dynamic-retrieval settings ($k = 1, 3, 5$). All settings use the same evaluation subset and the same downstream

Table 18: Failure categories in Experiment 4.

Failure category	Trigger condition
`execution_error`	The pipeline fails at the execution layer, typically because incorrect slot values lead to invalid source selection, wrong dimensions, wrong filters, empty results, or failed query execution
`operation_error`	The predicted task type or abstract operation does not match the gold intent, so the system enters the wrong execution path
`regulation_miss`	For regulation-related queries, the returned regulation evidence does not cover the gold regulation ID required by the gold answer
`calculation_error`	The system reaches the correct task and evidence, but the final numeric result or compliance decision is still wrong
`answer_mismatch`	Earlier pipeline stages succeed, but the final structured answer still does not match the gold answer in structure or content

execution modules. This design ensures that the observed differences primarily reflect the effect of the intent-recognition strategy on end-to-end system performance.

In addition to the main overall comparison, we also report grouped end-to-end results by query complexity in the results section. This grouped analysis helps reveal where dynamic retrieval provides the largest downstream gains.

7 Results

7.1 Experiment 1: Public benchmark results on CLINC150

7.1.1 Main results

Table 19 reports the main results of Experiment 1 on the 750-query CLINC150 evaluation subset. Overall, P gives the strongest performance among the schema-grounded methods. At $k = 3$, it reaches 0.960 Accuracy and 0.960 Macro-F1, and its performance remains similar at $k = 5$. By contrast, B1 performs much worse, while B2 improves over B1 but still stays clearly below P. For reference, we also report ICL no-schema results to assess the added value of schema in this benchmark setting.

7.1.2 Analysis

The results show that dynamic retrieval is the main source of improvement on CLINC150. P $k = 5$ outperforms B1 by 21.7 percentage points in Accuracy and outperforms B2 $k = 5$ by 9.9 points. Even P $k = 1$ already reaches 0.941 Accuracy, which is higher than all B2 settings. This pattern is consistent with the difference between static and dynamic examples. Static examples are fixed for all test queries, so they may still be semantically far from the current input even when they are

Table 19: Main results of Experiment 1 on the CLINC150 evaluation subset. B1, B2, and P denote the schema-only, static few-shot, and dynamic-retrieval methods; k denotes the number of few-shot examples.

Method	k	**Accuracy**	**Macro-F1**
B1 (schema-only)	–	0.744	0.757
B2 (static few-shot)	1	0.844	0.848
B2 (static few-shot)	3	0.832	0.834
B2 (static few-shot)	5	0.863	0.865
P (dynamic retrieval)	1	0.941	0.942
P (dynamic retrieval)	3	0.960	0.960
P (dynamic retrieval)	5	**0.961**	**0.961**
ICL no-schema	1	0.911	0.909
ICL no-schema	3	0.959	0.958
ICL no-schema	5	0.960	0.960

diverse as a set. By contrast, P retrieves query-specific nearest neighbours from the training set and therefore provides more relevant label signals at inference time. This makes the classification task easier, especially in a 150-label setting.

At the same time, the comparison with ICL no-schema shows that the extra contribution of schema is limited on CLINC150. At $k = 3$, P reaches 0.960 Accuracy, while ICL no-schema reaches 0.959. The gap is only 0.001. A very similar pattern appears in Macro-F1. This is understandable in the CLINC150 setting, where many intent labels are already semantically transparent, such as `credit_score` or `book_flight`. In this case, dynamically retrieved examples already provide enough information for the model to infer the correct label, and the additional schema information provides only limited extra help.

The results also show that B2 does not improve monotonically with k. For example, B2 $k = 3$ is slightly worse than B2 $k = 1$. This suggests that adding more fixed examples does not necessarily provide better guidance. When the added examples are not closely related to the current query, they may introduce noise rather than support. A related pattern appears in B1. B1 injects the full schema but does not provide any examples, and its Accuracy is only 0.744, which is much lower than ICL no-schema $k = 1$. This suggests that full-schema zero-shot prompting is not sufficient in a large intent space. When all 150 intent descriptions are injected at once, the model must distinguish many semantically related labels without any in-context anchor, and the schema alone becomes difficult to use effectively.

7.1.3 Summary

Experiment 1 shows that the proposed retrieval-based prompting strategy works well on a standard public benchmark. In particular, the results consistently support the use of dynamic retrieval over zero-shot and static few-shot prompting. This provides an important methodological basis for the later enterprise experiments. At the

Table 20: Main results of Experiment 2 on the enterprise validation set. B1, B2, and P denote the no-retrieval, static few-shot, and dynamic-retrieval methods; k denotes the number of few-shot examples.

Method	k	Category Acc	Operation Acc	Routing F1	Slot F1	Slot Value Acc	Core EM	Avg Token
B1 (no retrieval)	–	0.966	0.865	0.176	0.670	0.728	0.079	7,767
B2 (static few-shot)	1	0.966	0.865	0.195	0.685	0.731	0.124	7,923
B2 (static few-shot)	3	0.989	0.865	0.431	0.687	0.756	0.124	8,101
B2 (static few-shot)	5	1.000	0.876	0.938	0.733	0.798	0.528	8,446
P (dynamic retrieval)	1	0.989	0.933	0.852	0.703	0.793	0.528	7,914
P (dynamic retrieval)	3	0.989	0.955	0.944	0.770	0.896	0.742	8,192
P (dynamic retrieval)	5	**1.000**	**0.966**	**0.955**	**0.778**	**0.924**	**0.809**	8,452

same time, because CLINC150 intent labels are already semantically transparent, this benchmark is less suitable for fully revealing the contribution of schema grounding. Its main role in this thesis is therefore to provide benchmark-level validation before the evaluation moves to the more challenging enterprise setting.

7.2 Experiment 2: Enterprise structured intent parsing results

7.2.1 Main results

Table 20 reports the main results of Experiment 2 on the 89-query enterprise validation set. Compared with Experiment 1, the pattern here is more task-oriented: the main differences do not lie only in category or operation prediction, but more clearly in routing, slot extraction, and Core EM. Among all settings, the proposed dynamic-retrieval method P gives the strongest overall results. In particular, P reaches 0.742 Core EM at $k = 3$ and 0.809 at $k = 5$, while also achieving large gains in Routing F1 and slot-related metrics. By contrast, B1 remains very weak on Core EM, and B2 improves only when the static example set becomes large enough.

7.2.2 Analysis

The results show that the main difficulty in the enterprise setting is not basic intent recognition alone, but structured executable interpretation. B1 already reaches 0.966 Category Accuracy and 0.865 Operation Accuracy, which suggests that the schema-grounded prompt is sufficient for basic semantic understanding. However, its Routing F1 is only 0.176 and its Core EM is only 0.079. This large gap shows that recognising the broad task type is not enough. The model must also identify the correct source combination and the key executable fields, and this is where the zero-shot setting breaks down most clearly.

B2 improves over B1, but its behaviour remains unstable and highly sensitive to the fixed examples. At $k = 1$ and $k = 3$, B2 reaches only 0.124 Core EM, which is still close to B1. Only at $k = 5$ does it rise to 0.528. Even then, the improvement

Table 21: Representative routing cases in Experiment 2.

Case	**Example query**	**Gold routing**	**B1**	**B2** $k = 3$	**P** $k = 3$
Single-source	"In 2025 Q3, what percentage of products in the Netherlands are USNC compliant?"	`["usnc_pns"]`	`[]`	`[]`	`["usnc_pns"]`
Multi-source	"From USNC dashboard, for product CUC code=X, can it be claimed as 'very low sodium' in EU regulation?"	`["usnc_pns", "regulations"]`	`["usnc_pns"]`	`["usnc_pns"]`	`["usnc_pns", "regulations"]`
Cross-source	"List the PNS status, USNC status, HFSS status and Nutri-Score of `[PRODUCT_CODE]` in Netherlands."	`["usnc_pns", "hfss_6npm", "ns_6npm"]`	`[]`	`[]`	`["usnc_pns", "hfss_6npm", "ns_6npm"]`

comes with a clear increase in token cost. The gap between B2 $k = 3$ and B2 $k = 5$ is especially large, which suggests that static examples do not generalize reliably in this enterprise setting. Their usefulness depends heavily on whether the selected examples happen to cover the routing and slot patterns needed by the current query.

P shows the clearest overall improvement. At $k = 1$, it already reaches 0.528 Core EM, which matches B2 $k = 5$ with fewer prompt tokens. At $k = 3$, it reaches 0.742 Core EM and 0.944 Routing F1, and at $k = 5$ it further improves to 0.809 Core EM. The most important change, however, appears in routing. Compared with B1, P $k = 3$ improves Routing F1 by 0.768. Compared with B2 $k = 3$, the routing gain is still 0.513. This pattern suggests that the main advantage does not come from using more examples, but from using more relevant examples.

This also makes the role of schema more visible than in Experiment 1. In the enterprise setting, the intent space is less transparent, the data sources are heterogeneous, and the output is structured rather than label-only. Under this condition, schema grounding provides the task structure and the executable field space, while dynamic retrieval provides query-specific examples within that structured space. The improvement is therefore not only semantic, but also operational: the model becomes better at producing outputs that can support downstream routing and execution.

To make the routing difference more concrete, Table 21 shows three representative cases. These cases cover single-source, multi-source, and cross-source queries. Across all three cases, the main difference is not broad task understanding, but recovery of the correct source combination.

The table shows that B1 and B2 often identify the broad task correctly but still fail to recover the full routing pattern. In the single-source case, both methods predict an empty routing set. In the multi-source case, they recover only the dashboard source and miss the regulation source. In the cross-source case, they fail again, while P $k = 3$ recovers all three dashboards. These cases make the main gain of dynamic retrieval more visible: it helps the model recover source-combination patterns that are difficult to infer from the query alone.

A similar pattern appears in the subgroup analyses. On the query-complexity split, B1 and B2 remain weak on more difficult queries, while P $k = 3$ reaches

Table 22: Token-efficiency analysis in Experiment 2. Efficiency is defined as the Core EM gain divided by the additional prompt tokens relative to B1.

Method	k	**Core EM**	**EM Δ**	**Token Δ**	**Efficiency**
B1 (no retrieval)	–	0.079	baseline	baseline	–
B2 (static few-shot)	1	0.124	+0.045	+156	0.288
P (dynamic retrieval)	1	0.528	+0.449	+146	**3.075**
B2 (static few-shot)	3	0.124	+0.045	+334	0.135
P (dynamic retrieval)	3	**0.742**	**+0.663**	+424	**1.562**
B2 (static few-shot)	5	0.528	+0.449	+678	0.663
P (dynamic retrieval)	5	**0.809**	**+0.730**	+685	**1.066**

Table 23: Results by query-complexity label in Experiment 2. For readability, the multi-step and cross-source queries are grouped together as a single complex subset.

Subgroup	**N**	**B1**	**B2** $k = 3$	**P** $k = 3$	**Metric**
simple	81	0.074	0.111	0.716	Core EM
complex	8	0.125	0.250	1.000	Core EM

perfect Core EM on the complex subset. These results are consistent with the routing analysis above: once source combinations and structured conditions become more specialized, dynamic retrieval becomes much more important.

The token-efficiency results also support the choice of P $k = 3$ as the recommended setting. Table 22 shows that P $k = 1$ has the highest efficiency score, but P $k = 3$ provides a much stronger absolute Core EM while still keeping the extra token cost moderate. Compared with P $k = 5$, it gives up only a limited amount of performance while using fewer tokens. This makes P $k = 3$ the most balanced setting for later experiments.

Table 23 shows the same pattern from a subgroup perspective. The gain of dynamic retrieval is especially clear on more difficult queries. On the complex subset, P $k = 3$ reaches perfect Core EM, while B1 and B2 $k = 3$ remain much weaker. This is consistent with the routing analysis above: once source combinations and structured conditions become more specialized, dynamic retrieval becomes much more important.

The robustness pattern is clearer when the results are grouped by input variant, as shown in Table 24. B1 and B2 $k = 3$ both drop to zero Core EM on paraphrase and typo variants, while P $k = 3$ remains much more stable. The drop from orig to para is small for P, which suggests that dynamic retrieval is naturally more robust to semantic reformulation. Typo variants remain more difficult, but P still clearly outperforms the other settings. This again fits the overall picture of Experiment 2: the main difficulty lies not in broad intent understanding alone, but in structured executable mapping under enterprise-specific constraints.

Finally, the remaining-error analysis under P $k = 3$ shows that the dominant fail-

Table 24: Robustness across input variants in Experiment 2.

Method	Orig Core EM	Para Core EM	Typo Core EM
B1 (no retrieval)	0.091	0.000	0.000
B2 (static few-shot) $k = 3$	0.143	0.000	0.000
P (dynamic retrieval) $k = 3$	0.753	0.714	0.600

ures are no longer gross routing collapses. Instead, they shift toward finer-grained issues such as slot values, slot keys, and operation boundaries. This suggests that dynamic retrieval addresses the main routing bottleneck, while the remaining challenges lie in more detailed semantic normalization and structured prediction.

7.2.3 Summary

Experiment 2 is the central structured-intent experiment of this thesis. Overall, B1 shows that schema-grounded prompting is already sufficient for basic semantic understanding, but it remains weak on routing and therefore on executable structured prediction. B2 improves over B1, but its performance is unstable and strongly depends on the fixed examples. P gives the strongest and most consistent results, with the clearest gains in routing, Core EM, and robustness. Taken together, these results suggest that schema grounding provides the structural basis for enterprise intent parsing, while dynamic retrieval makes that structure much more effective at inference time.

Compared with the public benchmark in E1, the effect becomes much clearer here. On CLINC150, many intent labels are already semantically transparent, so the added value of schema grounding is limited. In the enterprise setting, however, the intent space is less transparent, the data sources are more heterogeneous, and the output must support downstream execution. Under this condition, schema grounding provides the task structure and executable field space, while dynamic retrieval provides query-specific examples within that structure.

Among all tested settings, P $k = 5$ gives the highest parsing score, but P $k = 3$ provides the best overall balance between structured accuracy, routing quality, robustness, and token cost. For this reason, we select P $k = 3$ as the default intent-recognition configuration for the later experiments. This also provides the basis for Experiment 4, where we examine whether the improvements in structured intent parsing transfer to the full end-to-end pipeline.

7.3 Experiment 3: Regulation retrieval results

7.3.1 Main results

Table 25 shows the main results of Experiment 3 on the 35-query regulation-retrieval set. Overall, the four methods form a clear performance hierarchy, especially at Hit@1: B1 performs worst, B3 improves substantially over B1, B2 further improves over B3, and P gives the strongest overall results. At $k = 1$, P reaches

Table 25: Main results of Experiment 3 on the regulation-retrieval set (35 queries).

Method	P@1	R@1	Hit@1	P@3	R@3	Hit@3	P@5	R@5	Hit@5
B1 (BM25)	0.257	0.193	0.257	0.305	0.669	0.800	0.251	0.886	0.971
B2 (rule-based)	0.857	0.674	0.857	0.457	0.940	0.971	0.286	0.957	0.971
B3 (BM25+SBERT)	0.714	0.555	0.714	0.390	0.850	0.943	0.263	0.924	1.000
P (hybrid)	**0.971**	**0.769**	**0.971**	**0.467**	**0.950**	**0.971**	**0.297**	**1.000**	**1.000**

Table 26: Grouped Hit@1 comparison between B2 and P in Experiment 3.

Grouping	Subgroup	N	B2 Hit@1	P Hit@1	ΔHit@1
Gold-answer structure	single_gold	22	0.818	0.955	+0.136
Gold-answer structure	multi_gold	13	0.923	1.000	+0.077
Variant ambiguity	conflict	11	0.545	0.909	+0.364
Variant ambiguity	no_conflict	24	1.000	1.000	+0.000

0.971 Precision, 0.769 Recall, and 0.971 Hit rate. At $k = 5$, it reaches 1.000 Recall and 1.000 Hit rate. Taken together, these results show that the performance gain in Experiment 3 comes from the gradual addition of lexical matching, semantic similarity, structured slot constraints, and slot-aware reranking.

7.3.2 Analysis

The clearest pattern in Table 25 appears at Hit@1:

$$\text{B1} < \text{B3} < \text{B2} < \text{P}.$$

B3 improves over B1, which shows that semantic similarity is already useful even without structured slots. B2 then improves over B3, which indicates that structured slot constraints contribute more than pure semantic similarity in this task. Finally, P improves over B2, which shows that slot-aware reranking still adds value after the candidate space has already been constrained by rules.

A more specific observation is that the gain of P appears mainly at Top-1 ranking. Compared with B2, P improves Hit@1 from 0.857 to 0.971, while the differences at $k = 3$ are much smaller and the two methods are already very close at Hit@3. This suggests that the main role of the semantic reranker is not to expand the candidate space, but to rank the correct regulation higher within the rule-recalled set.

To better understand where this gain comes from, we further group the queries in two ways. First, we distinguish between *single_gold* and *multi_gold* queries, depending on whether the gold regulation set contains one or multiple correct entries. Second, we distinguish between *conflict* and *no_conflict* cases. Here, a conflict case refers to a query for which the rule-recalled candidate set still contains mutually competing regulation variants, such as `general` vs. `meals_main_dishes` or `solids` vs. `liquids`.

Table 26 shows that the gain of P is concentrated mainly in conflict cases. In these queries, the rule-based retriever already finds a plausible candidate set, but the rules alone do not reliably distinguish among mutually compatible variants. This is

Table 27: Overall end-to-end results in Experiment 4 (n=86).

Method	k	**N**	**Pass**	**Pass Rate**	**Op Err**	**Exec Err**	**Reg Miss**	**Calc Err**	**Ans Mis**
No retrieval	–	86	76	88.4%	1	5	0	1	3
Static few-shot	1	86	77	89.5%	0	5	0	1	3
Dynamic retrieval	1	86	82	95.3%	0	3	0	0	1
Static few-shot	3	86	81	94.2%	0	1	0	1	3
Dynamic retrieval	3	86	**85**	**98.8%**	0	1	0	0	0
Static few-shot	5	86	83	96.5%	0	1	0	0	2
Dynamic retrieval	5	86	85	98.8%	0	1	0	0	0

where semantic reranking helps most. The single_gold/ multi_gold split shows that P improves over B2 in both settings. The improvement is moderate for single_gold queries and smaller but still positive for multi_gold queries, which suggests that the proposed reranker remains stable when multiple gold entries are acceptable. Overall, these grouped results suggest that the main contribution of P lies in resolving fine-grained ambiguity inside an already correct rule-defined candidate space.

7.3.3 Summary

Experiment 3 shows that the regulation retriever benefits from combining rule-based filtering with slot-aware semantic reranking. The main results form a clear progression from lexical retrieval to semantic retrieval, then to slot-aware retrieval, and finally to slot-aware semantic reranking. At the same time, B2 already provides a strong rule-based baseline, which means that the gain of P should be understood as a further improvement on top of an already accurate candidate-filtering stage. This gain comes mainly from better Top-1 selection rather than broader recall, and it is most visible in conflict cases.

7.4 Experiment 4: End-to-end pipeline results

7.4.1 Main results

Table 27 reports the overall end-to-end results on the 86 automatically evaluated validation queries. Dynamic retrieval gives the strongest overall performance. In particular, dynamic retrieval reaches 98.8% pass rate at both $k = 3$ and $k = 5$, while the no-retrieval baseline reaches 88.4%. Static few-shot also improves as k increases, but it remains below the dynamic setting.

This result shows that the gain observed earlier in Experiment 2 transfers to the full pipeline. In Experiment 2, the main improvement of dynamic retrieval appears in routing and executable structured prediction. In Experiment 4, the same effect appears as a reduction in end-to-end failures, especially execution-related failures. At the same time, the fact that dynamic $k = 3$ and $k = 5$ reach the same overall pass rate further supports the use of $k = 3$ as the default configuration.

Table 28: End-to-end pass rate by query-complexity label in Experiment 4.

Method	k	**Simple (n=81)**	**Complex (n=5)**
No retrieval	–	91.4%	40.0%
Static few-shot	1	91.4%	60.0%
Dynamic retrieval	1	96.3%	80.0%
Static few-shot	3	97.5%	40.0%
Dynamic retrieval	3	**98.8%**	**100.0%**
Static few-shot	5	98.8%	60.0%
Dynamic retrieval	5	98.8%	100.0%

7.4.2 Analysis

The no-retrieval baseline remains relatively strong in this experiment, but this is consistent with the evaluation setting rather than contradictory to Experiment 2. Most automatically evaluated queries are simple, and many belong to operations with relatively fixed answer structures. In addition, no retrieval here means that no few-shot examples are added; the shared task instructions, schema-grounded prompt structure, and output constraints remain unchanged. This is also consistent with Experiment 2, where the no-retrieval setting already performs reasonably on broad category and operation recognition but remains much weaker on routing and executable structured prediction.

The advantage of dynamic retrieval becomes clearer when the results are grouped by query complexity, as shown in Table 28. On simple queries, the differences among methods are relatively small. On complex queries, however, the difference is larger: the no-retrieval baseline passes only 2 out of 5 complex queries, while dynamic retrieval passes all 5 complex queries from $k = 3$ onward. This is consistent with the Experiment 2 findings, where the largest gain of dynamic retrieval also appears in routing-heavy and execution-sensitive queries. At the same time, the complex subset contains only five samples, so this pattern should be interpreted with some caution.

This also clarifies the relation between Experiment 3 and Experiment 4. Experiment 3 shows that the regulation retriever itself benefits from slot-aware semantic reranking under oracle-slot input. In Experiment 4, the regulation module is fixed to the hybrid configuration selected from Experiment 3 across all compared settings, so that the main variable remains the intent-recognition strategy. Under this controlled setting, the end-to-end gain can be attributed more directly to the improvement in intent parsing and routing.

7.4.3 Summary

Experiment 4 shows that the gain of dynamic retrieval transfers clearly from structured intent parsing to full end-to-end system performance. The benefit is most visible on complex queries, where dynamic retrieval reaches full pass rate from $k = 3$ onward. Since dynamic retrieval at $k = 3$ already reaches the same overall pass rate as $k = 5$, the choice of $k = 3$ as the default configuration is further supported

at the end-to-end level.

Overall, the four experiments support the rationale and advantages of the proposed method in the enterprise setting. The method proposed in this thesis is a schema-grounded retrieval-augmented framework for enterprise multi-source question answering. Its core idea is to use structured schema to constrain the task space, dynamic retrieval to improve prompting, and controlled execution modules to support downstream question answering.

Experiment 1 first validates the effectiveness of retrieval-augmented prompting on a public benchmark, showing that dynamic retrieval already brings stable gains in a general intent recognition setting. Experiment 2 then shows that, in the enterprise setting, the advantage of this method becomes much clearer, especially in routing, structured slot parsing, and executable structured prediction. This suggests that the combination of schema grounding and dynamic retrieval is better suited to enterprise queries, where intent names are less transparent, data sources are heterogeneous, and outputs are more structured. Experiment 3 further validates the design of the regulation retrieval module. Structured slot constraints first define the correct candidate space, and slot-aware semantic reranking then improves Top-1 selection within that space. This shows that the hybrid design of rule-based filtering and semantic disambiguation is effective in the enterprise regulation setting. Finally, Experiment 4 shows that these intermediate improvements do not remain only at the module level, but also transfer to end-to-end system performance, with the clearest gains appearing on more complex enterprise queries.

Taken together, the results show that the main advantage of the proposed method in the enterprise setting is not only higher accuracy at the module level, but also more stable handling of complex queries and better executability and reliability of the final answers.

8 System implementation

8.1 Prototype overview

We implement a prototype enterprise question-answering system for nutrition and health compliance analytics. The system consists of a frontend and a backend. The frontend provides the user interface and renders structured results, while the backend performs structured intent parsing, routing, source-specific execution, answer synthesis, and trace logging. The prototype connects natural-language questions to Power BI dashboards, local CSV sources, and a structured regulation resource. Its goal is not to build a general chatbot, but to validate the proposed schema-grounded orchestration framework in an executable enterprise setting.

Figure 4 shows the implemented prototype interface. The screenshot illustrates the main user-facing workflow: a user submits a natural-language question, and the system returns a grounded answer together with a dynamically generated chart and execution metadata.

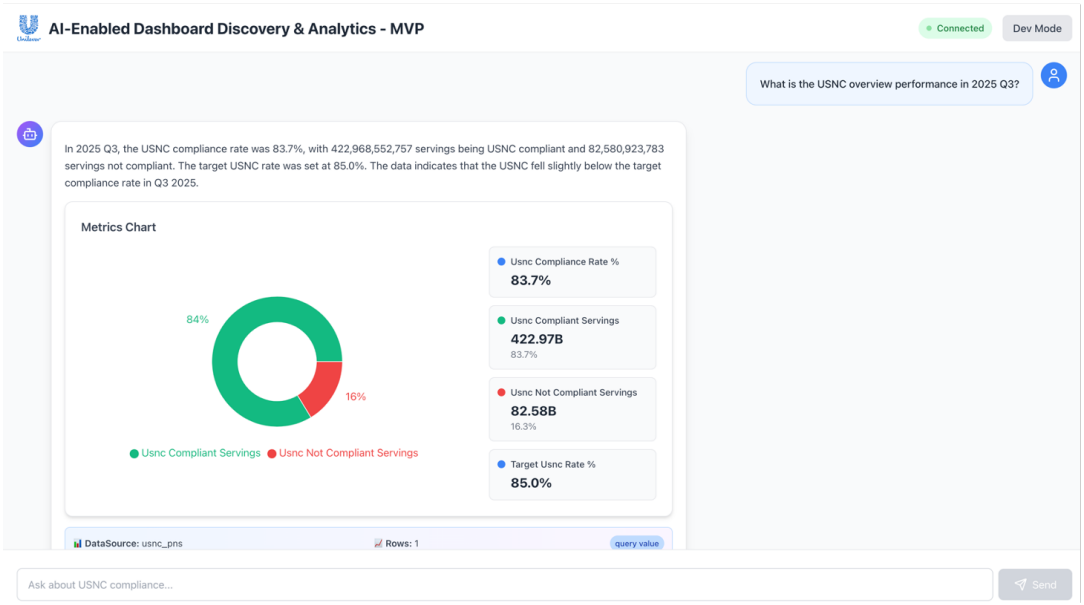


Figure 4: User interface of the implemented prototype system. The user submits a natural-language dashboard query and receives a grounded answer with a dynamically generated chart, selected data source, row count, and detected operation type.

8.2 Frontend implementation

The frontend is implemented with React and Vite. It provides a chat-based interface where users can enter natural-language questions, view the conversation history, and receive system answers. The interface follows the interaction style of common LLM-based chat systems, but it is adapted to enterprise analytics by displaying structured results together with the textual answer.

The frontend mainly reads the structured response returned by the backend. This response can include the final answer, selected data source, operation type, returned records, chart hints, regulation sources, compliance results, and trace identifiers. Based on these fields, the frontend dynamically renders different result formats, such as tables, KPI cards, charts, and compliance-check cards. Therefore, the frontend does not perform the main analytical logic itself. It focuses on user interaction and result presentation, while the backend performs intent parsing, routing, execution, calculation, and answer synthesis. Additional examples of supported user questions and corresponding output formats are provided in Appendix E.

In development mode, the frontend also displays intermediate execution information returned by the backend, including the parsed intent, generated DAX query, dashboard query results, and execution traces. Each execution step can be expanded to inspect its detailed output. This helps users to check whether the generated query is correct and whether the returned data is consistent with the final answer and visualization. This function is mainly used for debugging, error analysis, and system validation. Figure 5 shows the development-mode trace view for the example query shown in Figure 4.



Figure 5: Development-mode trace view showing intermediate execution steps for the query shown in Figure 4.

8.3 Backend implementation

The backend is implemented with FastAPI and provides the API layer between the frontend, the LLM service, and the connected data sources. A user query is sent from the frontend to the backend through the unified chat endpoint. The backend then processes the request through a modular pipeline: intent recognition, source routing, slot normalization, query generation, source execution, answer generation, and trace logging.

The first stage is intent recognition. The IntentRecognizer builds a schema-grounded prompt, optionally injects dynamically retrieved few-shot examples, calls the LLM, and returns a structured intent JSON. This JSON contains the question category, abstract operation, candidate data sources, execution mode, extracted slots, and optional control fields. After that, the SchemaHelper and SlotNormalizer validate the predicted fields and normalize slot values using schema definitions, lexicons, fuzzy matching, and time formatting. This step makes the parsed intent executable by aligning user expressions with source-specific field names, enum values, metrics, and business filters.

Most queries are handled by a simple fixed pipeline. For dashboard queries, the backend selects the relevant Power BI source, composes an executable DAX query from the normalized intent, executes the query through the Power BI connector, and normalizes the returned table. For CSV-based queries, the backend performs equivalent filtering, aggregation, ranking, or lookup over local structured data. For regulation lookup queries, the RegulationRetriever retrieves relevant EU, US, or Codex regulation entries from the structured regulation resource. For compliance-check queries, the ComplianceCalculator compares dashboard or user-provided nutrient values against the retrieved regulation thresholds and returns a structured eligibility result.

For complex queries, the backend switches to a planner–executor–aggregator flow. This mode is used when the parsed intent indicates that orchestration is required, such as cross-source analysis, dashboard-plus-regulation workflows, or multi-step analytical questions. The QueryPlanner decomposes the request into bounded executable steps, the StepExecutor runs each step against the relevant source-specific module, and the ResultAggregator merges intermediate results before answer gen-

eration. The planning process is constrained by the available schemas, supported operations, and execution templates, so the system does not allow arbitrary free-form tool use.

Finally, the AnswerGenerator produces the final natural-language response from grounded execution outputs and attaches structured payloads such as data previews, chart hints, regulation sources, compliance results, and a trace identifier. The TraceLogger stores execution traces in PostgreSQL, including the parsed intent, selected sources, generated queries, intermediate results, and execution metadata. These traces support debugging, evaluation, and inspection of the final answer.

9 Discussion

9.1 Overall implication of the results

Overall, the results show that the framework works well for enterprise nutrition and health compliance QA. It improves the intermediate analysis steps needed for execution, such as intent recognition, operation prediction, source routing, and slot extraction. It also improves regulation retrieval when compliance evidence is needed. These gains lead to better end-to-end execution. This means that the system better connects nutrition scientists' questions with the internal data sources, tools, and regulation rules needed to answer them.

More broadly, our work addresses a less explored QA problem in enterprise nutrition compliance. It makes three main contributions. First, we define the task as multi-source QA, where expert questions are linked to enterprise data, regulation evidence, and executable tools. Second, we propose a schema-grounded framework that turns these questions into structured workflows and produces answers based on clear data and regulation evidence. Third, we introduce an internal evaluation dataset and a structured regulation resource to support systematic evaluation.

The following subsections discuss what the framework means for method design and practical use, and then examine its remaining limitations and trade-offs.

9.2 Methodological implications

The results suggest that the LLM is most useful as a structured interpretation component rather than as an unconstrained answer generator. This is important in our task because the answer must be connected to real data sources, regulation evidence, and executable tools. A fluent answer alone is not enough. The system first needs an intermediate representation that can guide routing, retrieval, querying, checking, and answer synthesis.

The main components of the framework support this design in different ways. Structured intent and operation abstraction make user questions executable by turning them into fields that the backend can use. These fields include the data source, operation, execution mode, and slots. Dual schema grounding keeps these fields aligned with real enterprise sources, measures, field values, and regulation resources.

This reduces wrong source or field choices and makes the system easier to extend. Dynamic retrieval improves parsing by giving the model examples that are close to the current question. Slot-aware regulation retrieval improves evidence selection by using structured conditions before semantic matching.

Together, these designs make the analysis easier to execute. They help the system choose the right data source, extract the required parameters, select the right workflow, and produce an answer from grounded execution results.

9.3 Practical implications

The results have three practical implications.

First, the evaluation setting is close to real business needs. The enterprise evaluation set is based on interviews, questionnaires, and real internal business questions. Therefore, it reflects the daily analytical needs of nutrition scientists and product developers better than a general public QA benchmark. The dataset also has practical value because it turns scattered and implicit business questions into labeled QA tasks. This makes it possible to compare system versions and improve the pipeline around real workflows. The high end-to-end pass rate suggests that the framework supports many routine workflows represented in the current evaluation set.

Second, the framework has direct value for internal nutrition compliance workflows at Unilever. The connected dashboards and structured data sources allow the system to support dashboard analytics, cross-market comparison, and standardized compliance checking. For formulation improvement, product analysis, and multi-market claim assessment, scientists can check whether a product or formulation supports a specific claim and review the related nutrient values, thresholds, and market-specific rules. This reduces the time experts spend switching between dashboards, data tables, and regulation documents.

Third, the framework also helps with efficiency and checking. Lower prompt token usage reduces cost for frequent enterprise queries. At the same time, the system provides dashboard sources, regulation references, and execution traces. This allows experts to check where the answer comes from, instead of relying only on generated text. In practice, the framework does more than answer faster. It gives experts a more structured way to check and analyze nutrition compliance questions.

9.4 Limitations and trade-offs

This work has several limitations. First, the enterprise evaluation set is limited in size. It covers many routine and well-structured workflows, but it cannot represent the wider variety of questions that nutrition experts may ask. More diverse or less structured questions may affect structured interpretation, because the system still needs to map them clearly to the schema and available tools. Therefore, the current results mainly show how well the framework works within the studied setting. In future work, we need broader validation with naturally occurring user questions and longer-term usage logs.

Second, our experiments provide stronger evidence for the overall framework than for each backend module separately. We focus on the modules that are directly linked to the research questions, while other backend modules, such as query composition, DAX generation, threshold comparison, calculation logic, and result aggregation, are mainly evaluated through the full pipeline. It means that the end-to-end results show how well the system works as a whole, but they do not fully isolate the performance of each backend module or the exact source of each error.

Third, the system still depends on well-maintained schemas and regulation resources. This is a limitation because small changes in enterprise data can break execution. For example, BI and DAX queries require exact table and field names, while nutrition experts may use shorthand or dashboard-specific terms in their questions. The regulation resource also needs updates because the rules are stored as structured entries rather than extracted automatically from raw regulation texts. Therefore, when dashboard metrics, business definitions, or regulations change, we need to update and validate the schemas, field mappings, and regulation entries.

Finally, the framework supports expert analysis, but it does not replace expert judgment in compliance decisions. It is most useful for routine and well-structured workflows where the relevant data sources and regulation rules are already included. For high-risk or ambiguous cases, experts still need to check the evidence and final answer. The main trade-off is between open-ended flexibility and controlled execution. We do not let the LLM freely decide which tools to call. Instead, we define a limited task space from the observed enterprise questions, including question categories, operations, execution modes, and supported tool workflows. The LLM maps each user question into this structured space, and the backend executes the corresponding configured pipeline. This limits the range of questions the system can handle, but it makes execution more stable and easier to check.

10 Conclusion

In this thesis, we developed an executable and traceable multi-source question-answering system for enterprise nutrition and health compliance. The system provides one natural-language entry point to internal dashboards, structured data, and regulation resources. It brings these sources into one workflow and helps enterprise nutrition experts analyze compliance questions and make better-informed decisions.

To support this system, we proposed a schema-grounded orchestration framework. The framework represents user questions as structured intents and uses them to guide source routing, tool-backed execution, regulation retrieval, compliance checking, and answer synthesis. We evaluated it with an internal enterprise dataset and a structured regulation resource. The evaluation covers public benchmark validation, enterprise structured intent parsing, regulation retrieval, and end-to-end pipeline execution.

Taken together, the experimental results allow us to answer the three research questions as follows.

For **RQ1: How can natural language queries in enterprise nutrition and health compliance be represented as structured intents that support data source identification, tool selection, and executable workflow construction?**

In our framework, natural language queries are represented as structured intents with five main fields: question category, abstract operation, candidate data sources, execution mode, and slots. These fields turn an open user question into a task that the system can route, execute, and check. Experiment 2 shows that the system can predict the key intent fields, especially operation, source routing, and slots, which are needed for execution. Experiment 4 further shows that the parsed intents can drive the full pipeline. This means that structured intent works as a practical bridge between natural language questions and backend workflows.

For **RQ2: Under schema constraints, to what extent can schema-grounded prompting and dynamic few-shot retrieval improve structured intent parsing and downstream executability?**

Schema-grounded prompting with dynamic few-shot retrieval improves the execution-related parts of structured intent parsing, including operation prediction, source routing, slot extraction, slot value accuracy, and core structured matching. In Experiment 2, $k = 5$ gives the best parsing result, with Core EM increasing from 0.079 for the no-retrieval baseline to 0.809. However, Experiment 4 shows that $k = 3$ reaches the same end-to-end pass rate as $k = 5$ while using fewer prompt tokens. Therefore, $k = 5$ is the strongest parsing-only setting, while $k = 3$ is the more practical default setting for the full pipeline.

Overall, under schema constraints, dynamic retrieval gives a clear improvement in structured parsing and makes the parsed intents more useful for execution.

For **RQ3: To what extent can the proposed framework support correct and traceable end-to-end execution for dashboard analytics, structured data queries, and regulation-related tasks?**

Experiment 4 shows that the full pipeline reaches a 98.8% end-to-end pass rate. This means that the framework handles most routine end-to-end workflows represented in the current enterprise benchmark. During execution, the system records the selected data source, regulation evidence, tool outputs, and intermediate results. Experts can use these traces to check where the answer comes from and how it was produced. The framework works well for the current set of structured workflows, but more open or ambiguous workflows still need broader validation.

The results answer the research questions, but they also show where the framework needs further work. First, we need to test the system with more naturally occurring user questions and longer-term usage logs. This will show how well it handles less structured or more ambiguous questions. Second, we need to add clarification steps for incomplete questions. For example, when the user does not specify the data source, metric, market, or time period, the system should ask a follow-up question before execution. Third, we need to further reduce the maintenance effort for schemas, synonym mappings, and structured regulation resources. A simple di-

rection is to create a clearer update workflow for these resources. For example, when a dashboard field, business definition, or regulation rule changes, the corresponding schema, synonym mapping, and regulation entry should be reviewed and updated together. This would make maintenance more systematic and reduce the risk of outdated resources.

Appendix

A. User interview summary

User Interview Summary

We conducted three exploratory interviews with nutrition and health professionals at Unilever from different regions and roles. The participants are anonymized as P1, P2, and P3: an EU condiments manager, a North America nutrition capability leader, and an EU cooking aids product manager.

Main findings

- Users want to ask natural-language questions about product nutrition performance, compliance status, and possible nutrition claims.
- Current workflows are fragmented because users switch between dashboards, structured data sources, and regulation documents.
- Some tasks require calculation or rule interpretation, such as Nutri-Score query, HFSS analysis, and regional claim checking.
- Users need traceable answers that show the data source, regulation evidence, and calculation path.

Representative workflow needs

- Users want to ask questions such as compliance rate lookup by market, brand, product group, and period.
- Users often need to move from dashboard analysis to regulation checking, for example by identifying products that meet a nutrition standard and then checking whether they support a regional claim.
- Users may need a product-level multi-standard view. For example, for one anonymized product code, they may want to retrieve its USNC status, HFSS status, Nutri-Score, and related nutrition indicators from different dashboards.
- Users may also need to combine open-source nutrition datasets with internal product dashboards. For example, they may analyze zinc intake

or zinc deficiency rates by country or population group using WHO VMNIS data, and then compare these findings with zinc-related product information in internal dashboards.

- Some workflows require additional calculations or interpretation, such as RACC conversion, Nutri-Score or HFSS-related analysis, and threshold checking.

These findings informed the system design and the seed questions used for the enterprise 120-QA dataset.

B. Enterprise dataset and annotation examples

Representative Annotation Examples from the 120-QA Dataset

The full question set is not included because some questions and slot fields are based on internal product-related data and non-public market information. In addition to the core structured intent fields, the examples also show supporting execution fields, such as target metrics, output fields. These fields are used for query construction and answer verification, but they are not all direct prediction targets in the main structured intent evaluation.

B.1 Dashboard query example

```
Question:
What is the USNC compliance rate in Netherlands in 2025 Q3?

Gold structured intent:
{
  "question_category": "data_query",
  "abstract_operation": "query_value",
  "data_sources": ["usnc_pns"],
  "execution_mode": "simple",
  "slots": {
    "market": "Netherlands",
    "period": "2025 Q3"
  },
  "supporting_execution_fields": {
    "metrics": ["usnc_compliance_rate"]
  }
}
```

B.2 Cross-dashboard query example

```
Question:
List the USNC status, HFSS status, and Nutri-Score of product
[PRODUCT_CODE] in Netherlands.

Gold structured intent:
{
  "question_category": "data_query",
  "abstract_operation": "get_product_detail",
  "data_sources": ["usnc_pns", "hfss_6npm", "ns_6npm"],
  "execution_mode": "complex",
  "slots": {
    "cuc_code": "[PRODUCT_CODE]",
    "market": "Netherlands"
  },
  "supporting_execution_fields": {
    "output_fields": [
      "usnc_status",
      "hfss_status",
      "nutri_score"
```

```
    ]
  }
}
```

B.3 Regulation-related compliance check example

```
Question:
A soup contains 160 mg sodium per 100 ml. Does it qualify for a
"low sodium" claim in EU regulation?

Gold structured intent:
{
  "question_category": "compliance_check",
  "abstract_operation": "check_eligibility",
  "data_sources": ["regulations"],
  "execution_mode": "simple",
  "slots": {
    "product_group": "Soup",
    "claim_type": "low sodium",
    "nutrient": "sodium",
    "regulation_region": "EU",
    "nutrient_values": [
      {
        "nutrient": "sodium",
        "value": 160,
        "unit": "mg",
        "basis": "100ml"
      }
    ]
  }
}

Gold regulation target:
{
  "golden_regulation_id": ["eu_low_sodium/salt_liquids"]
}
```

C. Prompt templates

C.1 Structured intent parsing prompt template

```
System prompt:
You are an intent recognition system for an enterprise nutrition
compliance analytics platform. Parse the user question into a structured
intent JSON using the provided source registry, source schemas, and
few-shot examples.

Question categories:
- data_query: query dashboard or CSV data.
- compliance_check: check whether a product or nutrient value satisfies
  a nutrition claim condition.
- regulation_lookup: retrieve a nutrition claim standard or threshold.
- meta: system question or out-of-scope question.

Abstract operations:
- query_value: retrieve metric values.
- rank_by_dimension: rank entities by a metric.
- list_item: list products or records under filters.
- compare_values: compare values across dimensions.
- trend_over_time: analyze values over time.
- get_product_detail: retrieve product-level fields.
- check_eligibility: check claim eligibility.
- get_regulation_standard: retrieve a regulation rule.
- other: out-of-scope or unsupported query.

Runtime context:
```

```
The system first filters candidate sources using the global source registry.
It then inserts only the selected source schemas, including source descriptions,
slots, measures, metric groups, output fields, and guardrails. Finally, dynamic
few-shot retrieval selects query-specific examples and inserts them after the
schema context.

Dynamic examples:
{retrieved_few_shot_examples}

Key rules:
- Use only allowed categories, operations, and data sources.
- Use schema context to select sources, slots, measures, and outputs.
- Do not invent product codes, markets, nutrients, claim types, or regions.
- Put metrics, output_fields, rank_dimension, top_k, order_by, and group_by
  at the top level, not inside slots.
- Return valid JSON only.

User prompt:
Analyze this query and return structured JSON:
Query: "{user_question}"

Output format:
{
  "question_category": "<data_query | compliance_check | regulation_lookup | meta>",
  "abstract_operation": "<operation_name>",
  "data_sources": ["<source_id>"],
  "slots": {
    "period": null,
    "market": null,
    "region": null,
    "brand": null,
    "product_group": null,
    "nutrient_values": []
  },
  "metrics": null,
  "output_fields": null,
  "rank_dimension": null,
  "top_k": null,
  "order_by": null,
  "group_by": null,
  "requires_regulation": false,
  "requires_calculation": false,
  "execution_mode": "<simple | complex>"
}
```

C.2 Grounded Answer Generation Prompt Template

```
System prompt:
You are a professional nutrition compliance data analyst. Answer the
user's question based only on the retrieved execution results.

Dynamic context inserted at runtime:
- User Question: {query}
  The original user question.

- Market/Region: {market}
  The market, region, or regulation system inferred from the parsed intent.

- Available Data Summary: {data_availability}
  A short summary of which sources returned data, which sources failed,
  and whether regulation evidence or compliance calculation is available.

- Retrieved Data: {formatted_data}
  The formatted evidence produced by the execution modules. Depending on
  the question, this may include dashboard query results, CSV query results,
  retrieved regulation entries, compliance calculation results, or
  multi-source orchestration results.

- Structure Hint: {structure_hint}
```

```
  A short answer-format instruction selected according to the recognized
  intent, executed data sources, and answer type. For example, dashboard
  and CSV queries focus on retrieved values and concise insights;
  regulation lookup focuses on thresholds and official references;
  compliance checks include product values, regulation thresholds,
  calculation or comparison, and a pass/fail conclusion; cross-source or
  multi-step queries indicate which result comes from which source.

Key rules:
- Do not fabricate, infer, or assume numbers, percentages, or regulation facts.
- Use only values present in the retrieved data or calculation results.
- If a requested value is missing, state that it is not available in the
  retrieved results.
- For dashboard or CSV queries, report the retrieved value and concise insight.
- For regulation lookup, show the relevant threshold and official reference.
- For compliance checks, show the product value, regulation threshold,
  calculation or comparison, and final pass/fail conclusion.
- For cross-source or multi-step results, indicate which finding comes from
  which source.
- Return a clear natural-language answer.
```

C.3 CLINC150 auxiliary schema construction prompt template

```
Web-based ChatGPT prompt:
You are an intent schema construction assistant. Please read the
official CLINC150 train split JSON file I provide. The file contains
training utterances grouped by original intent label, together with
domain metadata. I want to construct an auxiliary schema for
schema-grounded intent classification.

Task:
For each original CLINC150 intent label in the file, generate one draft
schema entry.

Rules:
- Cover all original intent labels in the provided file.
- Keep the original intent labels unchanged.
- Do not create, merge, rename, or remove intent labels.
- Use the training utterances to infer the main meaning of each intent.
- Write one short natural-language description for each intent.
- Generate three to four representative keywords or trigger phrases based
  on the training utterances.
- Make the description and keywords representative of the intent and
  discriminative against similar intents.
- Return compact JSON only.

Output format:
{
  "label": "<original_intent_label>",
  "domain": "<domain_name>",
  "description": "<short description of the intent>",
  "keywords": ["<keyword_1>", "<keyword_2>", "<keyword_3>"]
}
```

C.4 Regulation text extraction prompt template

```
Web-based ChatGPT prompt:
You are a nutrition regulation data assistant. Please read the Excel
file I provide, which contains existing tabular nutrition claim rules,
and compare it with the relevant source regulation excerpt from the
official legal document.

Provided material:
- Existing Excel entry with region, claim type, food category, nutrient,
  threshold value, unit, and basis
- Source regulation excerpt from the official legal document
```

```
Task:
For each selected rule, extract the original legal wording and prepare
a draft structured regulation entry. The draft will be manually reviewed
before use.

Rules:
- Use the Excel entry as the main source for the structured rule fields.
- Use the source regulation excerpt to verify and extract regulation_text.
- Preserve the original legal wording as much as possible.
- Do not invent thresholds, units, nutrients, conditions, or references.
- If solids and liquids have different thresholds, create separate entries.
- Use id format: <region>_<claim_name>_<food_category>.
- Return compact JSON only.

Example output:
{
  "id": "eu_low_energy_liquids",
  "region": "EU",
  "claim_name": "LOW ENERGY",
  "variants": ["LOW ENERGY", "LOW CALORIE"],
  "food_category": "liquids",
  "regulation_text": "<original legal wording>",
  "threshold_text": "<=20 kcal/100ml",
  "reference": "(EC) No 1924/2006 - Annex: LOW ENERGY",
  "structured_rules": {
    "nutrient": "Energy",
    "operator": "<=",
    "threshold_value": 20.0,
    "unit": "kcal",
    "per_amount": "100ml",
    "condition_type": "Absolute"
  }
}
```

D. Schema examples

It provides simplified and anonymized examples of the two schema layers used in the system. The examples are shortened for readability. Internal dashboard connection identifiers, full field mappings, complete metric lists, and product-level identifiers are omitted.

D.1 Global data source registry example

Simplified global data source registry entry

```
data_sources:
  ns_6npm:
    source_id: "ns_6npm"
    type: "powerbi"
    name: "Nutri-Score Dashboard"

    description: >
      Dashboard for Nutri-Score analysis, including grade distributions,
      product-level score information, and portfolio-level summaries.

    schema_path: "schemas/ns_schema.yaml"

    routing_cues:
      - "Nutri-Score"
      - "NS score"
      - "grade A to E"
      - "front-of-pack score"
```

```
    supported_operations:
      - query_value
      - compare_values
      - rank_by_dimension
      - list_item
      - get_product_detail

    join_keys:
      - "product_code"

    connection:
      type: "powerbi"
      workspace_id: "[REDACTED]"
      dataset_id: "[REDACTED]"
```

D.2 Source-specific schema example

Simplified and anonymized source-specific schema excerpt

```
dataset:
  id: "ns_6npm"
  source_type: "powerbi"
  description: >
    Source-specific schema for Nutri-Score dashboard query generation.
    This excerpt is simplified and anonymized for illustration.

time:
  period:
    canonical_field: "[PERIOD_FIELD]"
    role: "filter_or_grouping"
    examples: ["2025 Q1", "2025 Q2"]

dimensions:
  market:
    canonical_field: "[MARKET_FIELD]"
    role: "filter_or_grouping"
    examples: ["Netherlands", "Germany"]

  region:
    canonical_field: "[REGION_FIELD]"
    role: "filter_or_grouping"
    examples: ["Europe", "North America"]

  brand:
    canonical_field: "[BRAND_FIELD]"
    role: "filter_or_grouping"
    examples: ["[BRAND_A]", "[BRAND_B]"]

  product_code:
    canonical_field: "[PRODUCT_CODE_FIELD]"
    role: "product_lookup"
    example: "[PRODUCT_CODE]"

  ns_score:
    canonical_field: "[NUTRI_SCORE_FIELD]"
    role: "filter_grouping_or_output"
    allowed_values: ["A", "B", "C", "D", "E"]

slots:
  allowed_filter_slots:
    - period
    - market
    - region
    - brand
    - product_code
    - ns_score
```

```
  allowed_group_by_fields:
    - period
    - market
    - region
    - brand
    - ns_score

measures:
  - name: "ns_grade_a_servings"
    role: "metric"
    unit: "servings"

  - name: "ns_grade_e_servings"
    role: "metric"
    unit: "servings"

default_output_fields:
  get_product_detail:
    - product_code
    - brand
    - market
    - ns_score

guardrails:
  require_valid_slot: true
  require_valid_measure: true
  allow_only_schema_fields: true
```

E. Anonymized system output examples

It provides 4 anonymized examples of prototype system outputs. Product identifiers, internal product names, brand names, market names, and sensitive values are anonymized where needed.

Example query: Compare the USNC compliance rate between China and Netherlands in 2024 Q3.



Figure 6: Dashboard comparison output example. It illustrates a single-source analytical query where the system retrieves a dashboard metric for two selected markets, compares the values, and returns both a visual result and a short analytical summary.

Example query: Show the PNS compliance rate trend for [BRAND] in EU from

2023 Q3 to 2025 Q3.

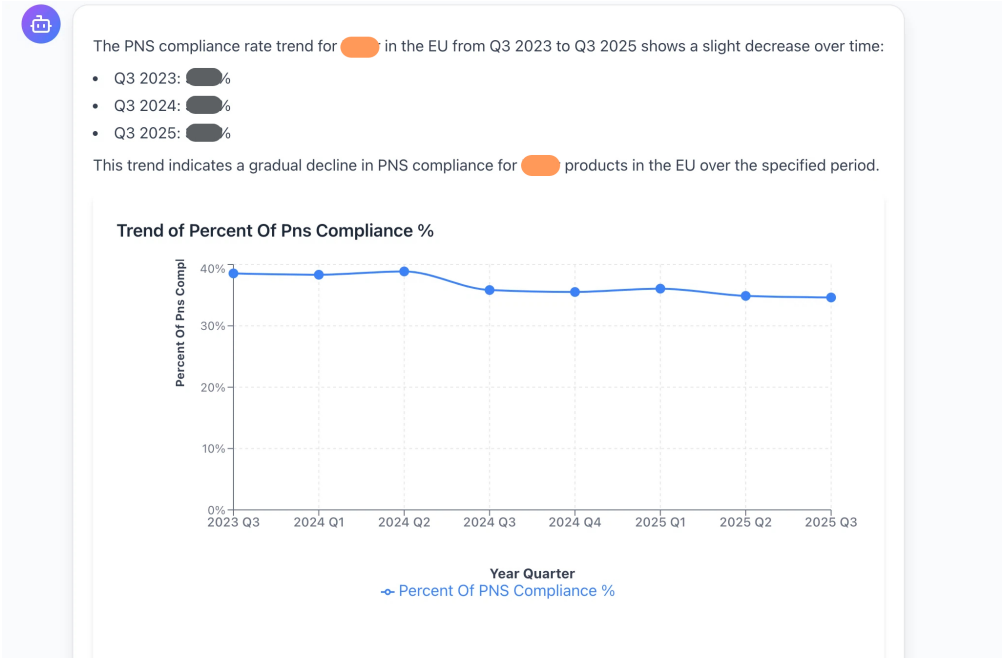


Figure 7: Time-trend output example. It illustrates how the system handles a `trend_over_time` query by retrieving period-based values and generating a concise trend interpretation.

Example query: What are the USNC compliance rate and PAM recyclability rate for [BRAND]?

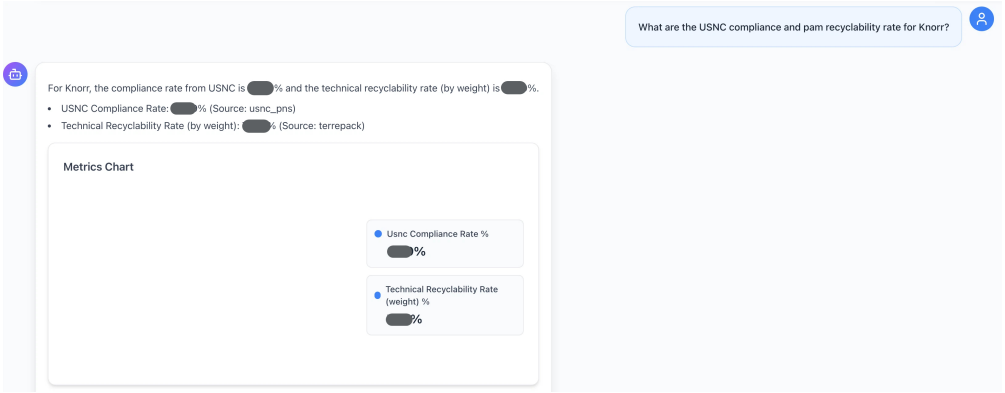


Figure 8: Cross-source output example. It illustrates a multi-source query where the system retrieves results from more than one source, aligns them under the same query scope, and states which result comes from which source.

Example query: A soup label shows sodium = 60 mg/100 ml. Can it be claimed as "low sodium" in the EU?

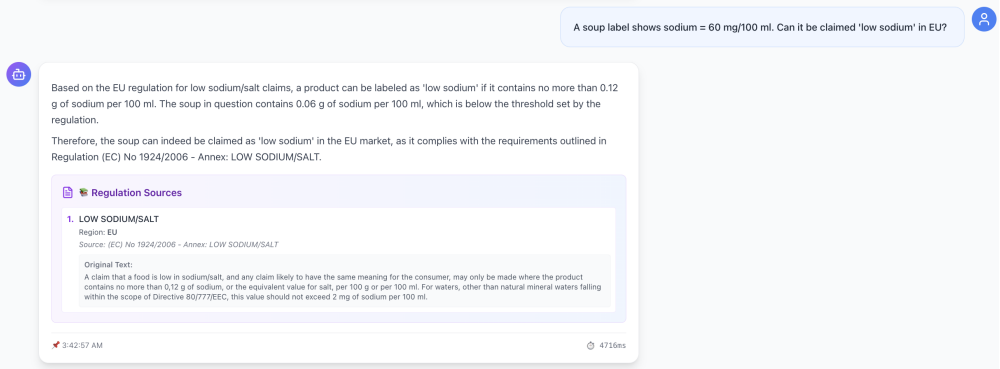


Figure 9: Regulation-related compliance output example. It illustrates how the system retrieves a structured regulation rule, compares a user-provided nutrient value with the threshold, and returns a conclusion with an evidence reference.

References

[1] Jinqing Lian et al. "Chatbi: Towards natural language to complex business intelligence sql". In: *arXiv preprint arXiv:2405.00527* (2024).

[2] Xinyu Liu et al. "A Survey of Text-to-SQL in the Era of LLMs: Where are we, and where are we going?" In: *IEEE Transactions on Knowledge and Data Engineering* (2025).

[3] Jamal Belkhouribchia and Joeri Jan Pen. "Large language models in clinical nutrition: an overview of its applications, capabilities, limitations, and potential future prospects". In: *Frontiers in Nutrition* 12 (2025), p. 1635682.

[4] Annalisa Szymanski et al. "Integrating expertise in llms: crafting a customized nutrition assistant with refined template instructions". In: *Proceedings of the 2024 CHI Conference on Human Factors in Computing Systems*. 2024, pp. 1–22.

[5] Zheyuan Zhang et al. "Ngqa: a nutritional graph question answering benchmark for personalized health-aware nutritional reasoning". In: *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 2025, pp. 5934–5966.

[6] Ana Gjorgjevikj et al. "Large language models in food and nutrition science: Opportunities, challenges, and the case of FoodyLLM". In: *Current research in food science* 12 (2026), p. 101351.

[7] Patrick Lewis et al. "Retrieval-augmented generation for knowledge-intensive nlp tasks". In: *Advances in neural information processing systems* 33 (2020), pp. 9459–9474.

[8] Shunyu Yao et al. "React: Synergizing reasoning and acting in language models". In: *International Conference on Learning Representations (ICLR)*. 2023.

[9] Jivitesh Jain, Nivedhitha Dhanasekaran, and Mona Diab. "From Complexity to Clarity: AI/NLP's Role in Regulatory Compliance". In: *Findings of the Association for Computational Linguistics: ACL 2025*. 2025, pp. 26629–26641.

[10] Islam Aushev et al. "RAGulator: Effective RAG for regulatory question answering". In: *Proceedings of the 1st Regulatory NLP Workshop (RegNLP 2025)*. 2025, pp. 114–120.

[11] Jaewoong Kim, Minseok Hur, and Moohong Min. "From rag to qa-rag: Integrating generative ai for pharmaceutical regulatory compliance process". In: *Proceedings of the 40th ACM/SIGAPP Symposium on Applied Computing*. 2025, pp. 1293–1295.

[12] Ohad Rubin, Jonathan Herzig, and Jonathan Berant. "Learning to retrieve prompts for in-context learning". In: *Proceedings of the 2022 conference of the North American chapter of the association for computational linguistics: human language technologies*. 2022, pp. 2655–2671.

[13] Aristides Milios, Siva Reddy, and Dzmitry Bahdanau. "In-context learning for text classification with many labels". In: *Proceedings of the 1st GenBench Workshop on (Benchmarking) Generalisation in NLP*. 2023, pp. 173–184.

[14] Jithinraj Edakkanambeth Varayil et al. "Artificial intelligence in clinical nutrition: Bridging data analytics and nutritional care". In: *Current nutrition reports* 14.1 (2025), p. 91.

[15] Mi-Young Kim et al. "Legal information retrieval and entailment using transformer-based approaches". In: *The Review of Socionetwork Strategies* 18.1 (2024), pp. 101–121.

[16] Jingyun Sun, Zhongze Luo, and Yang Li. "A compliance checking framework based on retrieval augmented generation". In: *Proceedings of the 31st International Conference on Computational Linguistics*. 2025, pp. 2603–2615.

[17] Ruben Kruiper et al. "A platform-based Natural Language processing-driven strategy for digitalising regulatory compliance processes for the built environment". In: *Advanced Engineering Informatics* 62 (2024), p. 102653.

[18] Paolo Falcarin et al. "Legal Requirements Compliance using NLP and Knowledge Graphs". In: *2025 IEEE 33rd International Requirements Engineering Conference Workshops (REW)*. IEEE. 2025, pp. 412–418.

[19] Henry Weld et al. "A survey of joint intent detection and slot filling models in natural language understanding". In: *ACM Computing Surveys* 55.8 (2022), pp. 1–38.

[20] Mohammadreza Pourreza and Davood Rafiei. "Din-sql: Decomposed in-context learning of text-to-sql with self-correction". In: *Advances in neural information processing systems* 36 (2023), pp. 36339–36348.

[21] Stephen Robertson and Hugo Zaragoza. *The probabilistic relevance framework: BM25 and beyond*. Vol. 4. Now Publishers Inc, 2009.

[22] Nils Reimers and Iryna Gurevych. “Sentence-bert: Sentence embeddings using siamese bert-networks”. In: *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (EMNLP-IJCNLP)*. 2019, pp. 3982–3992.

[23] Tao Yu et al. “Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task”. In: *Proceedings of the 2018 conference on empirical methods in natural language processing*. 2018, pp. 3911–3921.

[24] Stefan Larson et al. “An evaluation dataset for intent classification and out-of-scope prediction”. In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*. 2019, pp. 1311–1316.