Simplifying Complex Insurance Product Management with AI

João Teixeira[1][1201399], Paulo Gandra de Sousa[1], Luiz Faria[1], Duarte Cardoso[1], and Francisco Oliveira[1][1201545]

Instituto Superior de Engenharia do Porto mail@isep.ipp.pt
https://www.isep.ipp.pt/

Abstract. The digital transformation of the insurance sector presents significant challenges in internal processes, particularly in the management of product information. These challenges arise from the high complexity and variability of insurance product models, which are frequently updated, and from the technical demands of existing tools that require extensive user expertise to operate effectively. Recent advances in Generative Artificial Intelligence (GenAI) and the growing use of Large Language Models (LLMs) and intelligent agents are opening up new opportunities to automate and streamline processes, enabling organizations to adapt to a rapidly evolving technological landscape.
The Product Machine Explorer leverages Generative AI, Large Language Models (LLMs), and AI agents to make exploring insurance product models easier. Integrated with msg Life Iberia's Product Machine platform, it allows users to interact with complex product data using natural language. By leveraging structured data and advanced query techniques, it can understand user requests and deliver accurate, context-aware responses, improving both efficiency and usability in product model exploration.
Built using the Evaluation-Driven Development (EDD) methodology and supported by software, prompt, and context engineering, the tool was assessed using defined metrics and expert feedback. The results demonstrate significant efficiency improvements, with professionals spending considerably less time on repetitive tasks. Overall, the Product Machine Explorer demonstrates how LLM-powered agents can simplify complex information management and support better decision-making in the insurance sector.

Keywords: Insurance Products · Model Querying · Generative AI · Large Language Models · AI Agents · Agentic Workflows

1 Introduction

1.1 Context

Over the past years, the insurance industry has gone through a major digital shift that has changed how companies handle information, develop products, and

interact with customers. This shift has brought new challenges, particularly in product management, where the complexity arises from multiple variations, interrelations, and specific rules that must be continually adapted to match market demands. Even though tools exist to centralize information on product structures, detailed configuration and exploration of each product are still performed manually, leading to time-consuming, unintuitive, and operationally inefficient processes.

1.2 Problem

The growing complexity of insurance products and the increasing demand for faster data retrieval remain major challenges for companies striving to maintain competitiveness. Although platforms such as Product Machine - one of msg Life Iberia's products - centralize insurance product information, internal inefficiencies persist due to the unintuitive model navigation process. This complexity requires significant user expertise in the platform's internal operations, ultimately reducing productivity and increasing time spent on repetitive tasks. Furthermore, product models often embed intricate business rules and exhibit significant heterogeneity across different clients, further amplifying the difficulty of managing and evolving these systems.

As a result, there's a growing demand for easier and more efficient ways to access, search, and manage insurance product data.

1.3 Objective and Methodology

Generative Artificial Intelligence (AI) offers significant potential to address these challenges by enabling advanced natural language understanding, translating user intent into structured data operations, and generating coherent natural language responses, thereby bridging the gap between human communication and structured data interaction. Therefore, the main goal of this project is the design, development, and implementation of an Artificial Intelligence agent capable of performing natural language queries on insurance product models within the Product Machine. Additionally, the project aims to integrate this agent into the main Product Machine platform, enabling its use in a real operational context and ensuring the practical relevance of the proposed solution.

To achieve this goal, the Evaluation-Driven Development (EDD) methodology is adopted. This approach relies on the systematic definition and application of evaluations (evals) to objectively measure the quality and consistency of the system's responses against a predefined set of reference results - the gold standard [1]. EDD shares similarities with Test-Driven Development (TDD) in traditional software engineering but distinguishes itself through its more dynamic nature and specific suitability for AI-based systems, ensuring continuous improvement and performance-driven evolution of the agent [1].

2 Related Work

Recent advances in Generative Artificial Intelligence (AI) and Large Language Models (LLMs) have significantly impacted natural language processing and structured data interaction. Transformer-based models, such as GPT-4, demonstrate strong capabilities in contextual understanding, reasoning, and text generation [3]. However, their application to structured data querying, particularly in specialized domains like insurance, requires careful prompt design, data representation, and management of domain-specific business rules [3].

Research in Text-to-SQL has shown that LLMs can translate natural language into executable queries, yet challenges remain in handling complex schemas, ensuring query correctness, and managing multi-step reasoning for large or highly interrelated datasets [2]. Parallel developments in multi-agent LLM architectures, using frameworks such as LangGraph, enable collaborative, stateful workflows that integrate reasoning, tool use, and stepwise query execution [2]. However, recent evaluations reveal a gap between benchmark performance and real-world applicability. While LLMs achieve high accuracy on simple SQL tasks, their performance drops sharply on complex analytical queries, often producing semantic inconsistencies and structural errors [6]. Studies also indicate that even high-performing models may struggle when transitioning from generic benchmarks to enterprise environments, partly due to misalignment or oversimplification in widely used datasets [7]. These findings underscore the difficulty of applying LLMs to enterprise-grade schemas and highlight the need for domain-aware reasoning and structured workflows.

In the context of insurance product modeling, data representation is critical. Models are typically stored in relational databases, which remain dominant in enterprise environments due to their maturity, integration capabilities, and practical applicability. While the use of Knowledge Graphs (KGs) have been proposed as an alternative for representing complex relationships and supporting semantic reasoning [4], most practical implementations rely on SQL as the core data structure for LLM-assisted querying and knowledge exploration, given the SQL-centric infrastructures predominant in enterprise environments.

3 Product Machine

The Product Machine (PM), developed by msg Life Iberia, is a modeling platform that centralizes insurance product information - including coverages, rules, and calculations - and supports integration with downstream systems. It simplifies product modeling, validation, calculations, and reporting, helping teams develop products more efficiently. Built on the Eclipse Modeling Framework (EMF) and based on Model-Driven Engineering principles, PM uses technologies like the Object Constraint Language (OCL) and Query/View/Transformation (QVT) to support structured modeling, enforce rules, and perform model-to-model transformations.

The core strength of Product Machine is its flexibility, allowing each client to create a **custom metamodel** and generate unique product structures tailored

to their unique needs. Adding to this versatility, **component relationships within a product structure are dynamic**, adapting to specific conditions and contexts. This complexity is managed through Composition Rules (CRs) and Composition Rule Variations (CRVs). CRs define parent-child relationships, including cardinalities and dependencies, while CRVs allow these relationships to vary according to context, such as policy type, geographic region, or customer attributes. Availability formulas and functional states further regulate component behavior and visibility, enabling precise, rule-driven product configurations. Such flexibility, however, poses significant challenges for consistent querying and exploration of client-specific models.

Typical components that integrate client models include:

- **Products:** Primary insurance offerings that define the structure and rules for a specific type of insurance.
- **Packages:** Sets of coverages and entities grouped within a product to facilitate management, typically associated with the insurance risk.
- **Coverages:** Specific types of protection provided by an insurance product, detailing what is covered.
- **Participants:** Entities associated with the contract. For example, a person or organization can be the policyholder - responsible for payment and management - while another may be the insured party.
- **Pricing/Premium Details:** Contain the calculations and logic required to determine premiums, representing the detailed values computed during the process.

Overall, the Product Machine provides a powerful and highly configurable framework for insurance product modeling, but its flexibility requires careful handling when querying and analyzing client-specific configurations.

4 Solution

4.1 Functional Requirements

Defining the system's functional requirements is essential to establish a foundation for evaluating its performance across the key elements of a typical insurance product model. They outline the key information the agent must accurately retrieve and interpret. There are seven primary functional requirements (**RF**), each addressing a specific aspect of information gathering and interpretation.

- **RF1:** Query information about insurance products and product families
- **RF2:** Query information about packages within products
- **RF3:** Query information about coverages and their associations
- **RF4:** Query information about premium components and details
- **RF5:** Query information about insurable objects
- **RF6:** Query information about composition rule variations (CRVs)
- **RF7:** Query information about premium calculation formulas

Together, these requirements ensure comprehensive coverage of the core entities and relationships within insurance product models, forming the basis for evaluating the agent's accuracy and effectiveness.

4.2 System Design

The Product Machine Explorer was designed with a modular and extensible architecture. It integrates multiple components that work together to manage agent workflows, user interaction, and data access. The system comprises four main elements:

1. Backend agent orchestration using the LangGraph framework.
2. Frontend interface developed with Streamlit for rapid prototyping and user interaction.
3. MariaDB database that stores structured product information.
4. API layer exposing defined endpoints, through which Product Machine version five (PM5) can invoke dedicated agent workflows.

Fig. 1 illustrates the overall system architecture.

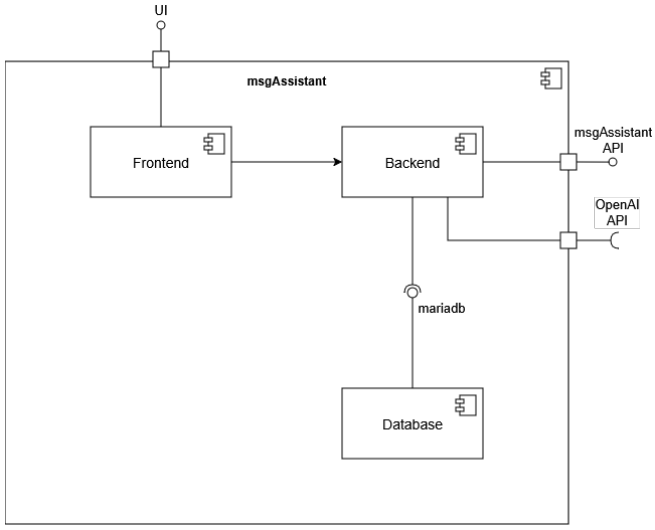

Fig. 1. System Design

This modular design ensures flexibility, ease of maintenance, and seamless coordination across all system components.

4.3 Data Structure Preparation

An essential step in the system's design was restructuring Product Machine's complex data model to enable efficient LLM-driven querying. This process involved:

- **View Creation:** Database views were implemented to encapsulate complex multi-table JOIN operations (often involving more than five tables) and to present unified logical groupings of data of component relationships.
- **Schema Simplification:** Column names in the view schemas were standardized and simplified to enhance LLM comprehension.
- **Composition Rule Variations (CRVs):** Special handling was implemented for CRVs, which represent conditional relationships between product components. Views were enriched to include availability formulas that determine when relationships apply.
- **Formula Extraction:** Premium calculation formulas were extracted from runtime-generated files and structured into dedicated tables, enabling queries about pricing logic.

4.4 Agent Workflow

The agent employs a directed acyclic graph (DAG) workflow consisting of nodes with edges that determine its execution flow, as illustrated in Fig. 2.

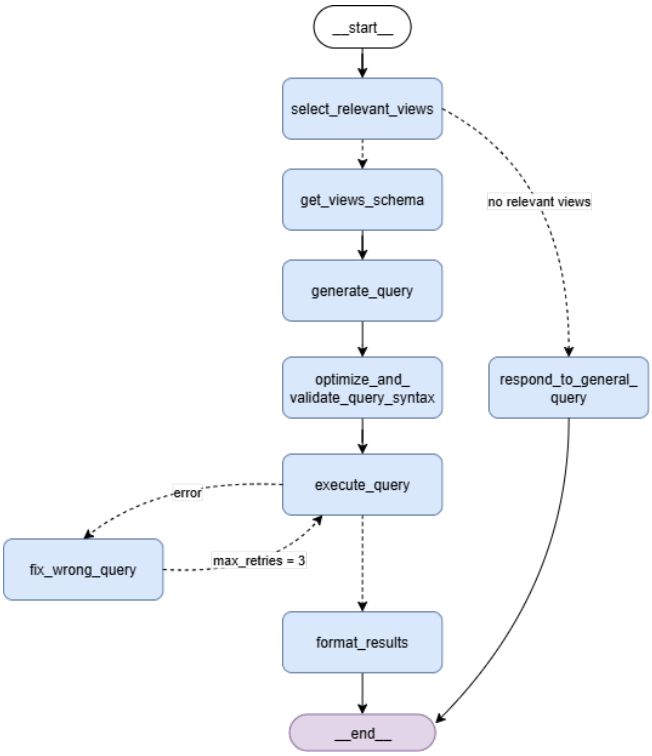

Fig. 2. Agent Workflow Diagram

This workflow consists of the following main components:

1. **View Selection:** Given a user query, the agent identifies relevant database views containing information needed to answer the question. This step employs few-shot prompting with examples of query-to-view mappings.
2. **View Schema Retrieval:** Extracts schema details for the selected views from the database to provide structural context for query construction.
3. **Query Generation:** The agent generates a SQL query based on the selected views and user intent. This step incorporates schema information, hierarchical relationship rules, and domain-specific constraints.
4. **Query Validation and Optimization:** The generated query is validated for syntax correctness and optimized for performance. Common SQL errors are identified and corrected, including improper clauses, missing JOIN conditions, and inefficient subqueries.
5. **Query Execution:** The validated query is run against the database with guard-rails in place to prevent any unauthorized operations. Only SELECT operations are allowed.
6. **Wrong Query Fix:** If a query fails, the system analyzes the error and tries to automatically fix it, allowing up to three retries.
7. **Result Formatting:** The query results are turned into clear, natural-language responses that directly answer the user's question, providing the right context and structure.

Additionally, a conditional path lets the agent handle user questions that aren't related to database operations, such as follow-ups or general questions about the Product Machine.

In short, this workflow creates a clear and reliable pipeline that turns natural language queries into accurate, optimized, and context-aware database responses, thus ensuring both operational reliability and real business value.

4.5 Prompt Engineering

Effective prompt engineering proved critical for system performance. Key techniques include:

- **Structured Instructions:** Prompts are organized hierarchically using Markdown formatting with clear sections for context, guidelines, and examples. In general, modern LLMs respond particularly well to well-structured prompts with specific sections and hints to direct the LLM [5].
- **Few-Shot Learning:** Critical nodes include examples that illustrate the correct query patterns for complex situations, especially when dealing with CRVs or navigating hierarchical structures.
- **Ordered Execution Steps:** For tasks that involve multiple steps, using ordered lists helps the model carry out each step in the correct sequence [5].
- **Domain Context:** Prompts provide detailed descriptions of insurance product structures, hierarchical relationships, and business rules. The provided context includes key terminology and conceptual frameworks from the insurance field to ensure accurate understanding and minimize ambiguity.

In addition to these techniques, a strategy was put in place to manage **multiple client-specific insurance models** that may vary in structure. Configuration dictionaries were introduced to capture instructions unique to each client that can't be applied universally. These dictionaries include database view mappings, component hierarchies, domain-specific terminology, and example queries, ensuring prompts are accurately adapted to each client's data.

5 Results

By adopting an Evaluation-Driven Development methodology, the system's iterative implementation was guided by the early definition of evaluation metrics designed to assess the agent's performance across diverse test cases. Two primary metrics form the basis of this evaluation:

- **M1 - Query Accuracy:** Average string similarity between generated and expected results must exceed 0.90 across all test cases.
- **M2 - Consistency:** Standard deviation of results across multiple executions must remain below 0.10, ensuring reliable performance.

These metrics ensure that development remains closely aligned with measurable performance goals throughout the system's lifecycle, promoting both accuracy and stability.

5.1 Quantitative Evaluation

Table 1 presents evaluation results for the final system iteration, demonstrating achievement of both defined metrics across all functional requirements.

Table 1. Agent Evaluation Results

Functional Requirement	Mean	Std Dev
RF1: Products	0.935	0.000
RF2: Packages	0.945	0.000
RF3: Coverages	0.960	0.000
RF4: Premium Components	0.965	0.000
RF5: Insurable Objects	0.985	0.000
RF6: Variations (CRVs)	0.958	0.015
RF7: Calculation Formulas	0.922	0.004
Overall Average	**0.953**	**0.003**

The system successfully exceeds the defined threshold (M1: 0.90) with an overall average accuracy of 0.953. Consistency metrics (M2) remain well below the 0.10 threshold, with most requirements showing zero deviation across five execution runs.

5.2 Qualitative Evaluation

Beyond quantitative metrics, the system was successfully integrated into the PM5 production environment and subsequently validated by insurance product specialists. Fig. 3 shows the newly added component.

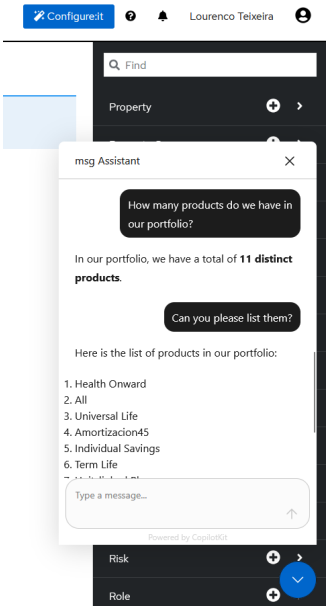

Fig. 3. Integration within PM5's interface

Technical users reported substantial efficiency improvements, as complex queries that previously required extensive manual analysis can now be executed in seconds. Moreover, it streamlines the retrieval of valuable product insights, demonstrating the system's practical value and robustness.

6 Conclusions

6.1 Key Findings

This work demonstrates that an SQL-based approach, supported by targeted knowledge engineering, is highly effective for querying insurance product models, leveraging SQL's proven maturity and enterprise-grade integration. Careful data structure design, through simplified naming, logical view grouping, and explicit business rule representation, significantly enhances system performance and query reliability. By combining language models of varying sizes, teams

can save costs without compromising performance, with lighter models handling simple tasks and more powerful ones tackling complex problems.

Finally, an evaluation-driven development process, grounded in continuous metric-based assessment, proved essential for iterative refinement and production-level reliability.

6.2 Limitations and Future Work

While the system demonstrates strong performance, several areas remain open for improvement. Currently, only the core product components within insurance product models are supported, and achieving full coverage will require refining context engineering and introducing purpose-built views.

Performance can also be enhanced, as complex queries with large results may exhibit latency. Future work could explore the use of query decomposition and result aggregation to enhance responsiveness.

Finally, evaluation methods rely on string-similarity metrics, which can unfairly penalize responses due to minor formatting differences and often require manual inspection of the results. Adopting LLM-as-a-Judge approaches could enable more reliable semantic evaluation of response quality, allowing assessments to capture correctness beyond superficial formatting differences and reducing the need for manual inspection [8].

References

1. Xia, B., Lu, Q., Zhu, L., Xing, Z., Zhao, D., Zhang, H.: Evaluation-Driven Development of LLM Agents: A Process Model and Reference Architecture. arXiv:2411.13768 [cs.SE] (2025). https://arxiv.org/abs/2411.13768
2. Zhu, X., Li, Q., Cui, L., Liu, Y.: Large Language Model Enhanced Text-to-SQL Generation: A Survey. arXiv:2410.06011 [cs.DB] (2024). https://arxiv.org/abs/2410.06011
3. Wang, B., Ren, C., Yang, J., Liang, X., Bai, J., Chai, L., Yan, Z., Zhang, Q.-W., Yin, D., Sun, X., Li, Z.: MAC-SQL: A Multi-Agent Collaborative Framework for Text-to-SQL. arXiv:2312.11242 [cs.CL] (2025). https://arxiv.org/abs/2312.11242
4. Huang, X., Ku, C.: Designing an Interpretable Question Answering System for Vertical Domains Based on Large Language Model and Knowledge Graph. In: Book Title or Collection Name (if applicable), pp. xx–xx. Publisher (2024). https://doi.org/10.3233/ATDE240503
5. OpenAI: GPT-4.1 Prompting Guide. OpenAI Cookbook (2024). https://cookbook.openai.com/examples/gpt4-1_prompting_guide
6. Choi, J.: Fact-Consistency Evaluation of Text-to-SQL Generation for Business Intelligence Using Exaone 3.5. arXiv:2505.00060 [cs.CL] (2025). https://arxiv.org/abs/2505.00060
7. Zhang, F., Zhou, R.: Refining Zero-Shot Text-to-SQL Benchmarks via Prompt Strategies with Large Language Models. Appl. Sci. 15(10), 5306 (2025). https://doi.org/10.3390/app15105306
8. Zeng, Q., Ma, S., Niknafs, A., Basran, A., Szabo, C.: Taming SQL Complexity: LLM-Based Equivalence Evaluation for Text-to-SQL. arXiv:2506.09359 [cs.DB] (2025). https://arxiv.org/abs/2506.09359