Bridging Business Language and Mathematical Optimization: A Controlled Semantic Layer for Constraint Specification

Gonçalo Nuno Ferreira Miller

Master's Dissertation

FEUP Supervisor: Prof. Xavier Andrade

Master in Industrial Engineering and Management

2026-06-25

Abstract

Retail personalization requires deciding which stimuli should reach which customers across large customer bases while satisfying multiple business restrictions, a decision complex enough to be treated as a constrained allocation problem. Optimization-based decision support is well suited to such problems, but it can only operate on formal, structured and validated inputs. Business users, however, express the restrictions that should govern an allocation in natural language, and translating those restrictions into the formal limits the optimization engine requires has typically involved manual technical mediation. This dependence introduces delay, reduces the autonomy of the users who best understand the business intent, and exposes the process to the risk that a formally valid limit may not reflect the intended meaning.

This dissertation develops a controlled semantic layer that connects business users to an existing optimization engine through a deliberate separation of responsibilities. A Large Language Model (LLM) interprets a restriction and proposes a candidate structured representation. A domain description grounds that interpretation in the available attributes and values, deterministic validation checks structural and data compatibility before execution, an assembly step constructs the corresponding formal limit, and the optimization engine, left unchanged, remains responsible for the allocation. The LLM intervenes only in the interpretation step. Control over what reaches execution rests on the deterministic components, so the interpretation is treated as a hypothesis to be verified rather than as a final constraint.

The solution was built and validated incrementally. A first end-to-end workflow was established as a working Minimum Viable Product (MVP) over a controlled subset of real data, and was then extended toward more demanding operating conditions, including support for more complex requests, execution over full-scale production data, and configuration-driven reuse in a different use case. Across the validation cases, supported restrictions were transformed into the validated structured form the optimization engine executes, while structurally invalid, unsupported or data-incompatible interpretations were prevented from reaching execution. The evidence gathered is therefore technical and functional. It supports the robustness and practical executability of the proposed layer. Measuring operational value, such as time saved, falls outside the scope of this work. The contribution lies not in applying a language model to optimization, but in a controlled architecture that makes the specification of business restrictions explicit, inspectable and controlled while preserving the rigor of the optimization engine.

Resumo

A personalização no retalho exige decidir que estímulos devem chegar a que clientes. Quando esta decisão envolve um grande número de clientes e múltiplas restrições de negócio, atinge um nível de complexidade suficiente para ser tratada como um problema de alocação com restrições. O apoio à decisão baseado em otimização é adequado a problemas deste tipo, mas requer informação formal, estruturada e validada. Os utilizadores de negócio, porém, exprimem as restrições que pretendem aplicar a uma alocação em linguagem natural, pelo que a tradução dessas restrições para os limites formais exigidos pelo motor de otimização dependia de mediação técnica manual. Esta dependência gera atrasos, reduz a autonomia dos utilizadores que melhor compreendem a intenção de negócio e expõe o processo ao risco de que um limite formalmente válido possa não refletir o significado pretendido.

A presente dissertação desenvolve uma camada semântica controlada entre os utilizadores de negócio e um motor de otimização existente, com base numa separação deliberada de responsabilidades. Um Modelo de Linguagem de Grande Dimensão (LLM) interpreta cada restrição e propõe uma representação estruturada candidata. A descrição do domínio orienta essa leitura com base nos atributos e valores disponíveis. A validação determinística verifica a compatibilidade estrutural e com os dados antes da execução. Uma etapa de montagem constrói o limite formal correspondente, e o motor de otimização, mantido inalterado, continua responsável pela alocação. O LLM intervém apenas na interpretação. O controlo sobre o que chega à execução pertence aos componentes determinísticos, sendo a interpretação tratada como uma hipótese sujeita a confirmação e não como uma restrição final.

A solução foi construída e validada de forma incremental. Numa primeira fase, foi estabelecido um Produto Mínimo Viável (MVP) com um fluxo completo, validado sobre um subconjunto preparado de dados reais. Esse fluxo foi depois alargado para responder a condições de utilização prática, incluindo o suporte a pedidos mais complexos, a execução sobre dados de produção à escala real e a reutilização, mediante configuração, num caso de uso diferente. Nos testes realizados, as restrições suportadas foram convertidas na forma estruturada e validada que o motor de otimização executa, e as interpretações estruturalmente inválidas, não suportadas ou incompatíveis com os dados foram impedidas de chegar à execução. A evidência obtida é, por isso, de natureza técnica e funcional. Sustenta a robustez e a executabilidade prática da solução. A medição do valor operacional, como o tempo poupado, fica fora do âmbito deste trabalho. O contributo não está em aplicar um modelo de linguagem à otimização, mas numa arquitetura controlada que torna a definição das restrições de negócio explícita, inspecionável e controlada, sem comprometer o rigor do motor de otimização.

Agradecimentos

Este momento marca o fim de um dos capítulos mais importantes e bonitos da minha vida. Foram anos de crescimento, de desafios, de descobertas e de memórias que guardarei sempre com carinho. Mais do que uma meta alcançada, este ciclo ensinou-me que a verdadeira beleza está no caminho: nas pessoas que nos acompanham, nas pequenas conquistas de cada dia e na forma como, sem nos apercebermos, vamos crescendo com tudo aquilo que vivemos.

Em primeiro lugar, gostaria de agradecer à MC Digital pela confiança depositada em mim e pela oportunidade de poder trabalhar num ambiente tão dinâmico e estimulante. À Inês, a minha orientadora na empresa, agradeço a disponibilidade, a paciência e todo o acompanhamento ao longo deste projeto. Ao Francisco e à Teresa, obrigado pela ajuda, pelas sugestões e por tudo o que me ensinaram. A toda a equipa, deixo também o meu sincero obrigado pela forma como me receberam e por cada momento de aprendizagem.

Gostaria também de agradecer ao meu orientador, o Professor Xavier Andrade, por toda a disponibilidade e rigor ao longo desta dissertação. O apoio que me deu foi fundamental para tomar melhores decisões e enfrentar este percurso com mais confiança e clareza.

À minha família, e em especial aos meus pais e aos meus irmãos, deixo o agradecimento mais profundo. Obrigado por serem o meu apoio constante e por fazerem com que cada conquista tenha mais significado. Sou um bocadinho de todos vocês: dos valores que me transmitiram, do exemplo que me deram e do amor com que sempre me acompanharam. Este percurso foi meu, mas também é vosso. Porque tudo o que alcancei foi sempre mais fácil, mais bonito e mais importante por vos ter a meu lado.

"Reality is greater than ideas."

Pope Francis

Contents

Acronyms

AI	Artificial Intelligence
API	Application Programming Interface
DMN	Decision Model and Notation
DSS	Decision Support Systems
IIS	Irreducible Inconsistent Subsystem
JSON	JavaScript Object Notation
LLM	Large Language Model
MVP	Minimum Viable Product
NLI	Natural Language Interface
NLIDB	Natural Language Interface to Databases
OR	Operations Research
SDG	Sustainable Development Goal
SQL	Structured Query Language
UI	User Interface

List of Figures

List of Tables

Chapter 1

Introduction

This opening chapter introduces the dissertation and the problem it addresses. It begins by motivating the work, situating retail personalization as a large-scale allocation decision that optimization-based systems can support but that remains difficult for business users to operate directly. It then describes the institutional context in which the project was developed, presents the problem and the research questions that guide the work, outlines the methodological approach taken, and previews the structure of the remainder of the document.

1.1 Motivation

Retail personalization has become a central activity for companies that interact with large customer bases across multiple channels. At its core, it involves deciding which stimuli — such as coupons, products, banners or content blocks — should reach which customers, so that commercial objectives are advanced while business rules are respected. When a single customer base, a wide catalog of possible stimuli, and a growing set of business rules are considered together, this becomes a complex allocation decision. The feasibility of the allocation depends on how assignments jointly satisfy eligibility constraints and business rules, so decisions that seem reasonable in isolation may be jointly infeasible.

Decisions of this kind belong to the domain of prescriptive analytics, the most action-oriented stage of analytics, which addresses not only what may happen but what should be done, and among whose principal methods is optimization (Lepenioti et al., 2020). Optimization-based decision support is well suited to allocation problems of this scale, because it can evaluate many interacting choices simultaneously against a defined objective rather than resolving them one rule at a time. Its effectiveness, however, depends on the quality of the inputs it receives: an optimization model is defined over decision variables, an objective function, and constraints. A restriction that is stated informally cannot be consumed by such a model until it has been expressed in a precise, structured and consistent form (Hillier and Lieberman, 2010).

This requirement creates a practical barrier. Business users naturally express business restrictions in natural language, in the terms in which campaign rules are discussed, whereas an

optimization engine requires formal, structured and validated inputs. Bridging this distance has typically demanded technical expertise that business users do not have, which keeps those who best understand the business intent dependent on specialists to translate it into a form the system can execute. Natural language has long been studied as a way to lower this barrier, and recent Large Language Models (LLMs) have broadened the range of interpretation tasks that can be addressed. Yet such models, while flexible, are not reliable on their own. They can produce fluent and plausible output that is nonetheless false (Lin et al., 2022) or unfaithful to the input (Ji et al., 2023). Their interpretations must therefore be grounded and verified rather than trusted directly. Making optimization-based decision support accessible to business users requires more than interpreting language. It requires transforming business restrictions into formal, validated and executable structures, under control, without compromising the rigor that the optimization engine demands.

1.2 Institutional Context

This dissertation was developed within MC Digital, the technological unit of MC Sonae, a major Portuguese retail group. MC Digital applies data science, artificial intelligence and software engineering to decision-making problems in retail, including personalization processes across physical and digital channels. This setting provided the applied context for the project, where analytical and optimization-based systems support large-scale retail decisions.

1.3 Project Description

Within this context, the allocation decisions underlying personalization are supported by an existing optimization engine, referred to as Maestro. Maestro treats personalization as a constrained allocation problem over candidate customer–stimulus pairs and is applied across personalization products such as mailings, personalized leaflets and newsletters. To execute, it operates only on structured inputs. These inputs include formal limits, which are the scenario-specific element representing the business rules that must hold for a given case.

The difficulty addressed by this dissertation lies in producing those limits. A business restriction stated in natural language must be translated into a formal limit that is structurally valid and compatible with the prepared data before it can be executed, and this translation has depended on manual technical mediation. This dependence introduces delay, reduces the autonomy of the business users who best understand the intent, and exposes the process to a subtler risk, since a limit can be formally valid while still encoding a meaning different from the one intended. The problem is therefore not the optimization engine, which already resolves the allocation, but the dependency on the manual bridge between business language and the formal limits the engine requires.

The project originated from the broader aim of making interaction with optimization systems more accessible to business users. Within that scope, this work focuses on the specification of business restrictions and develops a controlled semantic layer that mediates between business users

and the existing optimization engine. The objective is to make restriction specification explicit, inspectable and controlled, rather than dependent on manual technical translation, while keeping the optimization engine responsible for the formal allocation it already performs. The contribution of this work lies not in applying a language model to optimization, but in the controlled architecture in which it is placed: a separation of responsibilities in which the LLM interprets, deterministic components validate, and the optimization engine retains the allocation it already performs. This objective is examined through four research questions:

RQ1. What limitations characterize the current process for specifying business restrictions in an optimization-based retail personalization system?

RQ2. How can a controlled semantic layer be designed to connect natural-language business restrictions with formal optimization while preserving the responsibilities of the existing optimization engine?

RQ3. How can business restrictions expressed in natural language be transformed into valid Maestro-compatible limits within the supported scope?

RQ4. How can the developed solution be evaluated in terms of technical robustness and practical executability under realistic conditions?

1.4 Project Methodology

The work followed an incremental engineering approach, structured so that each stage built on a validated foundation. It began with an analysis of the existing restriction-specification process and of the engine's formal input contract, which identified the gap to be addressed and the requirements a solution would have to satisfy. From this analysis, the formal structure that a restriction must take to become executable was defined, fixing the target that the proposed solution would have to produce.

A controlled semantic layer was then designed around a separation of responsibilities, in which natural-language interpretation, structured representation, deterministic validation and formal execution are kept as distinct responsibilities. This design was implemented and validated incrementally: a first cycle established the complete workflow end-to-end as a working Minimum Viable Product (MVP), and successive extensions then adapted it to the conditions of realistic use by broadening the range of supported restrictions, scaling the workflow to full-scale production data, testing reuse in a second configured use case, and explaining outcomes in business terms. Throughout, the evidence gathered was technical and functional, drawn from validation over realistic restrictions rather than from a formal user study or a quantitative measurement of operational value. The work closes by discussing the limitations of what was demonstrated and the directions for future work that follow from them.

1.5 Dissertation Structure

The remainder of this dissertation is organized as follows. Chapter 2 reviews the literature that provides the theoretical background to the work, covering optimization-based decision support, the formalization of business restrictions, natural language interfaces, and the use of language models within controlled architectures, and identifies the research gap that the dissertation addresses. Chapter 3 describes the context and the problem in detail, framing retail personalization as a customer–stimulus allocation problem, presenting the existing optimization engine and the inputs it requires, and characterizing the gap between business language and formal limits. Chapter 4 presents the methodology and the design of the controlled semantic layer, including its architecture and the separation of responsibilities on which it rests. Chapter 5 reports the implementation, validation and results across the successive development cycles. Finally, Chapter 6 presents the conclusions, discusses the limitations and contributions of the work, and proposes directions for future research.

Chapter 2

Literature Review

The current chapter establishes the theoretical background that supports the work developed in this thesis. The review begins with a broad view of decision support and optimization-based decision-making, which provide the framing within which the dissertation is situated, and then narrows down to the formalization of business rules, which is the activity that makes those models usable by business users. From there, the review turns to the natural language interfaces and semantic parsing techniques through which language has historically been mapped into structured representations, and finally to LLMs and the validation mechanisms required to embed them in formal decision contexts. The chapter closes by identifying the research gap that this trajectory exposes and to which the dissertation responds.

2.1 Optimization-Based Decision Support

Research on Decision Support Systems (DSS) has long described interactive, computer-based environments that help decision-makers use data and models to address semi-structured and unstructured problems (Sprague, 1980). In this framing, a DSS is defined by the interaction among users, data and models rather than by any particular technology, and its purpose is to augment human judgment on problems that are not fully automatable.

A more specific strand concerns systems built around quantitative models. Power and Sharda (2007) characterize model-driven DSS as systems in which algebraic, decision-analytic, financial, simulation or optimization models are made accessible to non-technical specialists through a usable interface. Their account identifies two defining properties: the model is exposed so that a user can manipulate its parameters and conduct "what-if" analysis, and this interactive, repeated use distinguishes a model-driven DSS from a one-off analytical study performed by a specialist.

The position of optimization within this tradition is clarified by work on the maturity of analytics. Lepenioti et al. (2020) distinguish descriptive analytics, which addresses what has happened, predictive analytics, which addresses what may happen, and prescriptive analytics, which addresses what should be done and why. The same review characterizes prescriptive analytics as the most action-oriented stage, identifies optimization among its principal methods, and notes that

it spans a spectrum from decision support to decision automation. In decision support, the system recommends actions. In decision automation, the prescribed action is implemented directly. This progression is shown in Figure 2.1, where optimization-based decision support is positioned within the prescriptive stage.

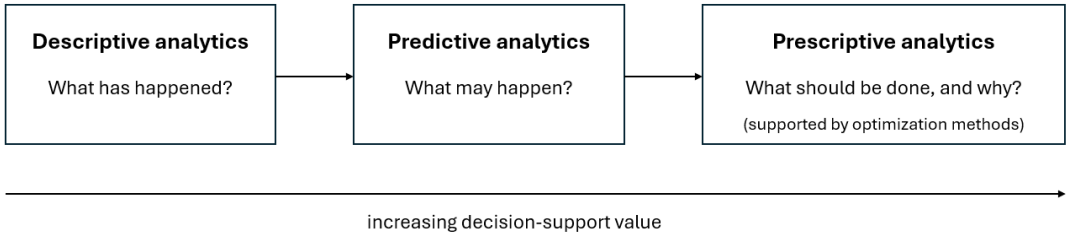


Figure 2.1: Evolution of business analytics maturity. Based on Lepenioti et al. (2020).

Optimization-based decision support depends on the quality of the inputs it receives. Operations Research (OR) provides the broader methodological context, spanning techniques such as mathematical optimization, simulation and queuing theory. The relevant branch here is mathematical optimization, in whose classical modeling sequence a real problem is defined and a mathematical model is formulated, solved, validated and implemented (Hillier and Lieberman, 2010). The formulation step expresses the decision in precise terms: an objective to be optimized, decision variables representing the available choices, and constraints delimiting the feasible region. In general form, such a model is written as in Equation (2.1).

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq b_i, \quad i = 1, \ldots, m, \\ & x \in X. \end{aligned} \tag{2.1}$$

In Equation (2.1), x is the vector of decision variables, f is the objective function, each inequality $g_i(x) \leq b_i$ is a constraint with bound b_i, m is the number of constraints, and X is the set of admissible values. The minimization form is generic, and maximization objectives are expressed analogously. As the formulation makes clear, an objective or a restriction stated informally cannot be consumed by an optimization model. Each constraint must be defined over specified variables and parameters, and a constraint set that is ill-formed or mutually inconsistent may render the model infeasible or yield a solution that does not reflect the intended decision. Allocation problems, in which limited resources are assigned across entities subject to capacity and eligibility constraints, are a canonical instance of this structured form (Hillier and Lieberman, 2010).

2.2 Business Rules and Constraint Formalization

Producing valid constraints from intent expressed in informal business language is, in essence, a problem of requirements formalization. Nuseibeh and Easterbrook (2000) describe requirements

engineering as the discovery of a system's purpose and of the needs and constraints of its stakeholders, together with the modeling, analysis, communication and validation of those needs in a form suitable for implementation. Their roadmap stresses the distance between requirements elicited through informal, contextual inquiry and the formal specification techniques that systems ultimately require. This distance is not purely linguistic, since converting a stated need into a precise, consistent constraint remains necessary even after the need has been faithfully transcribed.

The tension between accessible but ambiguous natural language and precise but specialist formal notation has motivated work on controlled natural language. Fuchs et al. (2008) present Attempto Controlled English as a restricted subset of English that remains readable by humans while mapping onto an unambiguous formal representation suitable for knowledge representation and reasoning, demonstrating that the gap between human-readable and machine-processable expression can be narrowed by disciplining the language admitted.

Once a business restriction has been represented formally, the representation itself can be analyzed. Calvanese et al. (2018) study decision tables in the Decision Model and Notation (DMN) standard, treating decision logic as a structured artifact whose correctness can be examined. Their algorithms detect missing rules, which violate completeness when some admissible input is left uncovered, and overlapping rules, which violate consistency when more than one rule fires for the same input. The same work shows that such inconsistencies can be detected within the representation itself, provided the representation is related to an explicit description of the domain, such as a schema, a controlled vocabulary or a lightweight ontology, against which terms can be mapped and validated.

2.3 Natural Language Interfaces and Semantic Parsing

This section reviews how natural language has been used as an entry point to data and models. It considers first natural language interfaces and their limitations, and then semantic parsing and the structured, grounded representations into which language can be mapped.

2.3.1 Natural Language Interfaces

Natural language has long been proposed as a means of lowering the technical barrier to data and model interaction, well before current language technologies. Androutsopoulos et al. (1995) review early Natural Language Interfaces to Databases (NLIDBs), built to let non-experts retrieve information without learning a formal query language, and report a recurring set of limitations: ambiguity, uneven and often opaque coverage, difficulty of porting across domains, and mismatches between what users expect the system to understand and what it can handle. These early systems relied largely on hand-crafted grammars and domain-specific rules, and although the field has since moved toward learning-based methods, the same tensions persist. Surveying Natural Language Interfaces (NLIs) to data, Quamar et al. (2022) emphasize that their value lies precisely in freeing non-technical users from formal query languages. Katsogiannis-Meimarakis and Koutrika (2023) observe that the alternatives remain unattractive to this audience, being either

form-based interfaces with limited expressiveness or low-level query languages accessible only to experts.

Quamar et al. (2022) further decompose natural language querying into three tasks: identifying the entities present in a user's utterance, connecting those entities over the underlying data source to interpret the user's intent, and generating a structured query as output. Abstracted from databases, this progression — interpretation, grounding, structured generation — characterizes any system that must turn language into a formal, executable representation.

2.3.2 Semantic Parsing and Grounding

Semantic parsing addresses the mapping of language into an explicit intermediate representation rather than directly into execution, a mapping that depends on the structure of the target domain rather than on language alone. Zettlemoyer and Collins (2007) study the learning of mappings from sentences to lambda-calculus logical forms, i.e., formal meaning representations that can subsequently be acted upon. Liang et al. (2013) separate interpretation from execution explicitly. A natural-language question is answered by representing its meaning as a logical form, and the answer is then computed by evaluating that logical form against a structured database of facts. This two-stage organization places a separable, inspectable artifact between a user's words and a system's actions.

A recurring finding is that robust interpretation cannot rest on linguistic analysis alone but requires grounding in the structure and vocabulary of the target domain. Popescu et al. (2004) demonstrate this with the PRECISE system, in which a statistical parser is treated as a modular component whose errors are corrected by a strong semantic model, so that questions deemed "semantically tractable" with respect to the underlying schema are mapped reliably. Domain constraints, in that account, actively restrict and repair the parser's output. The same concern recurs in work that maps text to Structured Query Language (SQL). Wang et al. (2020) define schema linking as the alignment of entity references in a question with the intended schema columns and tables, and treat it as central to generating valid queries. Yu et al. (2018), through the Spider benchmark, show that systems must generalize to previously unseen schemas and query structures rather than reproduce patterns memorized from training data in which the same database appears in both training and test sets.

The validity of a generated representation can also be enforced during generation. Scholak et al. (2021) observe that a language model fine-tuned to produce a constrained formal language frequently emits invalid output. Their PICARD method constrains decoding incrementally so that inadmissible tokens are rejected as they are generated, rejecting invalid candidates before they reach an execution engine.

2.4 Large Language Models in Controlled Architectures

The work reviewed above largely predates the adoption of large-scale neural language models, which have since reshaped both interpretation and structured generation. This section reviews

their capabilities and reliability limitations, and then the validation, oversight and governance mechanisms through which their outputs can be controlled.

2.4.1 Capabilities and Reliability Limitations

The methods reviewed above relied on hand-crafted grammars or on parsers trained for a specific schema. LLMs differ in that they interpret varied natural language without task-specific engineering, which broadens the range of interpretation and structured generation tasks that can be addressed. In exchange, they offer none of the structural guarantees those earlier methods built in, which motivates caution. Bommasani et al. (2021) describe the broad capabilities of foundation models alongside systemic risks, observing that the defects of a base model are inherited by every system adapted from it and that such models are best understood as sociotechnical. An architectural response is to combine language models with deterministic components rather than rely on the model alone. Toolformer illustrates this, training models to delegate tasks such as arithmetic and factual lookup to external tools through simple interfaces (Schick et al., 2023).

The reliability limitations are well documented. Lin et al. (2022), with the TruthfulQA benchmark, show that language models can produce fluent and plausible yet false answers, in some cases reproducing common human misconceptions, and that the largest models were not the most truthful. Ji et al. (2023) survey hallucination in natural language generation and distinguish intrinsic hallucination, in which generated output contradicts its source, from extrinsic hallucination, in which output introduces information that the source neither contains nor supports. Both failure modes matter where the output is a constraint: an intrinsic hallucination can contradict the stated request, and an extrinsic one can introduce assumptions that were never expressed, neither of which is removed by more fluent generation.

2.4.2 Validation, Oversight and Governance

Reliable individual outputs are necessary but not sufficient, because interpretation typically forms part of a user-facing workflow. Amershi et al. (2019) consolidate guidelines for human interaction with Artificial Intelligence (AI), holding, among others, that a system should make clear what it can do, convey the uncertainty of its outputs, explain how a result was reached, and support efficient correction. At the level of the system as a whole, the AI Risk Management Framework of the National Institute of Standards and Technology (National Institute of Standards and Technology, 2023) frames AI systems as sociotechnical, with risks arising from the interaction of technical components, human users and organizational context, and characterizes trustworthy systems in terms of properties including validity and reliability, transparency and accountability.

Table 2.1 brings together the principal challenges reported across these areas. The implications drawn in the final column are not claims of the cited sources but the requirements this dissertation derives from those challenges for controlled mediation between natural language and a formal optimization engine.

Table 2.1: Challenges reported by different sources in language-to-constraint mediation.

Challenge	**Sources**	**Implications**
Ambiguity and inconsistency of informal, natural-language statements	Androutsopoulos et al. (1995); Nuseibeh and Easterbrook (2000)	Interpretation requires clarification and consistency checking and cannot be accepted at face value
Vocabulary gap between user terms and formal model elements	Popescu et al. (2004); Wang et al. (2020)	Interpretation must be grounded in an explicit domain description that maps user terms to formal elements
Structurally or semantically invalid generated outputs	Scholak et al. (2021); Katsogiannis-Meimarakis and Koutrika (2023)	Candidate representations require deterministic structural and consistency validation before execution
Plausible-but-false interpretations (hallucination)	Lin et al. (2022); Ji et al. (2023)	The interpreting LLM should not be the final authority; outputs require external verification
Opaque or unilateral automation	Amershi et al. (2019); National Institute of Standards and Technology (2023)	The system should communicate its interpretation, expose uncertainty and support user confirmation and correction

2.5 Research Gap and Contribution

Read together, the streams reviewed in this chapter form a coherent picture. Optimization and prescriptive analytics establish the value of formal models and the requirement that their inputs be consistent. Requirements and business-rule research show how informal intent can be turned into checkable representations. Natural language interfaces and semantic parsing show how language can be mapped into a validated intermediate representation. The literature on language models shows that this mapping can be made flexible but is not, on its own, reliable. Figure 2.2 summarizes, at a conceptual level, the layered pattern these strands form when read together. Yet the specific integration this dissertation requires remains insufficiently addressed.

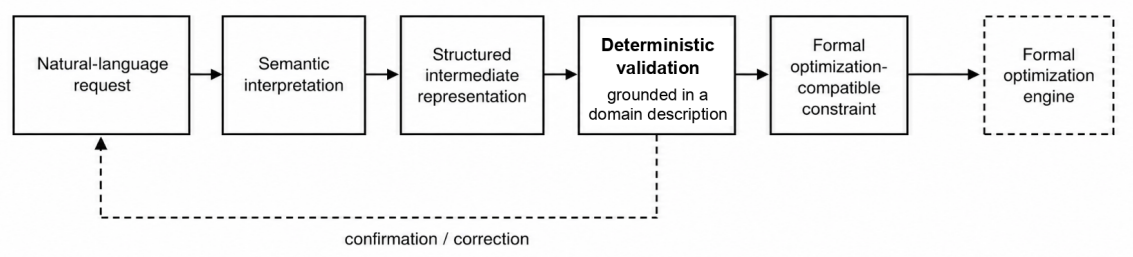


Figure 2.2: Layered mediation between natural language and formal optimization. Based on Quamar et al. (2022) and Scholak et al. (2021).

This limitation can be understood in three respects, each concerning the way natural-language interpretation is connected to formal optimization in decision-support settings.

First, the stakes of interpretation error differ across settings. Recent work applies LLMs to the end-to-end formulation of optimization models from natural-language problem descriptions: the NL4Opt competition defines this task as the translation of such descriptions into solver-compatible meaning representations (Ramamonjison et al., 2023), OptiMUS illustrates a system that formulates and solves mixed-integer linear programming problems from natural language (AhmadiTeshnizi et al., 2024), and Xiao et al. (2025) survey this stream as automatic optimization modeling with LLMs. In that work, the model itself is constructed from the text. This differs from optimization-based decision support over an existing model, where a misinterpreted natural-language request becomes a formal constraint added to that model. That misinterpretation can affect the validity of the recommended decision, shifting the burden of validation toward the formal correctness and consistency of the generated constraint. This is distinct from the feasibility of the constraint set as a whole, which is determined only when the model is solved.

Second, the optimization and decision-support literature has concentrated on formulating and solving models rather than on how business users specify and validate the constraints those models consume (Hillier and Lieberman, 2010; Power and Sharda, 2007). The upstream activity of turning business intent into valid, optimization-compatible constraints has therefore received comparatively little systematic attention.

Third, the reliability and governance literature prescribes that the outputs of LLMs be validated and overseen (Ji et al., 2023; National Institute of Standards and Technology, 2023). However, it does not specify how that validation should be organized when the required output is not free text but a constraint that must conform to the formal structure an optimization engine accepts.

This gap motivates the contribution of this dissertation: a controlled semantic layer in which semantic interpretation, structured representation, deterministic validation and formal execution are kept as distinct responsibilities. Within such a layer, an LLM contributes flexible interpretation, an explicit domain description grounds and constrains that interpretation, deterministic components validate the resulting representation, and the optimization engine remains solely responsible for optimization, while the user can inspect what was understood. The literature supports the plausibility and design rationale of this separation of responsibilities. Its practical value in a concrete decision-support setting is examined in the remainder of the thesis.

Chapter 3

Problem Description

Retail personalization in the company under study is supported by an existing optimization engine that operates only over formal inputs. The problem is the transformation of business restrictions expressed in natural language into limits that are valid and compatible with that engine.

This chapter develops the problem from the general decision context to the specific limit-specification process. It starts by framing retail personalization as customer–stimulus allocation, followed by the presentation of the existing optimization engine and the formal inputs required for execution. It then describes the current limit-specification process, from the expression of a business restriction to its technical translation, execution, review and possible adjustment. The analysis closes by identifying the main limitations and risks of this process, defining the concrete problem addressed in the dissertation.

3.1 Retail Personalization

This section begins by presenting the business context and then explores the logic behind personalization activities. Personalization is framed as a recurring operational decision with a constrained allocation structure, supporting the need for a formal optimization-based approach.

3.1.1 Business Context

Personalization is delivered through several distinct products, adapting what customers receive to their expected interests. In mailings, promotional coupons are selected according to individual preferences. In personalized leaflets, a weekly set of relevant promotional products is chosen for each customer. In newsletters, an email is assembled from banners or content blocks selected according to expected relevance. Although these products differ in channel and format, they share a common underlying decision: for each customer, a set of stimuli — coupons, products, banners or content blocks — must be selected and presented consistently across a large customer base and a wide range of possible stimuli.

This common decision is the relevant focus of the business context. The question is not whether personalization should be used, but how a limited and governed set of stimuli can be

assigned to customers in a way that advances commercial objectives such as relevance, engagement or expected value, while respecting business rules. In this sense, personalization is treated as a recurring, constrained decision rather than as a broad institutional or marketing context.

3.1.2 Customer–Stimulus Allocation

At the structural level, personalization is an allocation problem. On one side, there is a set of customers; on the other, a set of stimuli. In the modeled assignment space, each customer–stimulus combination is represented as a candidate assignment with an estimated value — an affinity or expected relevance. The objective is to select the combination of assignments that best satisfies the intended criteria. Not every candidate assignment is selected: depending on the objective and the applicable restrictions, a customer may receive none, one, or several stimuli. Figure 3.1 represents this framing at a conceptual level. For readability, only a subset of candidate assignment links is shown.

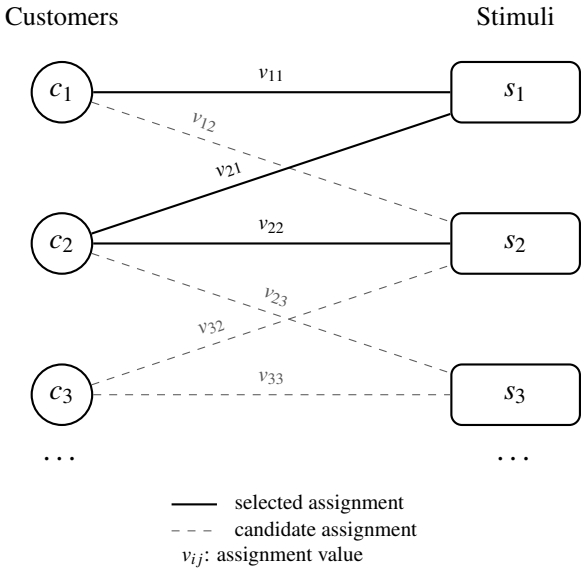


Figure 3.1: Customer–stimulus allocation in retail personalization.

The need for optimization arises from the restrictions that link decisions across the assignment space. Limited resources must be assigned across customers and stimuli subject to eligibility, capacity and other business rules, including requirements specific to each campaign. Some restrictions act on individual customers, such as a maximum number of stimuli a single customer may receive; others act on individual stimuli, such as a minimum or maximum audience for a given stimulus; and others apply to defined subsets, such as eligibility conditions that reserve certain stimuli for customers meeting a stated criterion. Because these restrictions interact, a stimulus assigned to satisfy one objective can consume capacity needed elsewhere, and assignments that appear attractive in isolation may be jointly infeasible.

A small example can make this interaction clearer. The values and specific assumptions are illustrative, but the situation reflects a real type of personalization scenario. Consider a personalized leaflet campaign covering mainland Portugal and the islands. Because the campaign is defined over the full national customer base, regional rules are expressed by singling out the customers of a given region. Each customer can receive at most 2 products. Within this campaign, the Madeira customers form one such regional group, comprising 100 customers. This creates 200 available assignment slots for that group. For business reasons, suppose that only 3 products are available to be assigned to customers in Madeira. To be commercially relevant, each of these products must reach at least 80 customers. Taken separately, each rule is reasonable: the per-customer maximum keeps the leaflet focused on the most relevant products for each customer, the regional eligibility condition reflects business relevance, and the minimum audience ensures that each product has sufficient reach. Taken together, however, the 3 minimum-audience requirements would require 240 assignments, while the Madeira group can provide only 200 slots. The issue is not that any single rule is invalid, but that valid rules interact globally. This conflict can be stated formally. Let $x_{cs} = 1$ when stimulus s is assigned to customer c, with the stimuli here being the campaign's products. Let M be the set of 100 Madeira customers and P_M the set of 3 products available to them. The per-customer maximum is expressed in Equation (3.1) and the per-product minimum audience in Equation (3.2).

$$\sum_{s} x_{cs} \leq 2 \quad \forall c \in M. \tag{3.1}$$

$$\sum_{c \in M} x_{cs} \geq 80 \quad \forall s \in P_M. \tag{3.2}$$

The per-customer rule in Equation (3.1) caps the group at $100 \times 2 = 200$ assignments, while the three minima in Equation (3.2) together demand $3 \times 80 = 240$. The gap between the two makes the set jointly infeasible.

The decision cannot be handled by considering each rule in isolation; all candidate assignments and restrictions must be evaluated together against a common objective. This simultaneity and combinatorial coupling make the allocation decision difficult to handle through manual selection or simple priority rules, motivating a formal, optimization-based approach.

3.2 Optimization Engine and Formal Inputs

The existing optimization engine supports the allocation decision, but it can only operate on structured data and formal limits. This section defines the role and scope of the engine in the personalization process and describes the input structures on which execution depends.

3.2.1 Role and Scope

At a conceptual level, Maestro treats personalization as a constrained allocation problem over candidate customer–stimulus combinations. It represents each candidate assignment as a decision variable, encodes restrictions as constraints, and selects the combination of assignments according to a defined objective. Mailings, personalized leaflets and newsletters share this decision structure, allowing the engine to be applied across these personalization products.

At the execution level, Maestro receives structured input tables and formal limit definitions that already describe the allocation problem: the candidate customer–stimulus combinations, their associated values, the attributes needed to define relevant subsets, and the restrictions to be enforced. From these inputs, the engine creates the decision variables, objective function and constraints, invokes a mathematical solver, and returns the optimized assignments together with a technical execution status. This status indicates whether the solver proved a solution optimal, returned a feasible solution without proving it optimal, or found the model infeasible.

The scope of the engine is limited to this formal execution stage. It is not responsible for collecting or preparing business data, estimating the values associated with candidate assignments, or interpreting restrictions expressed in natural language. These elements must be prepared and formalized before reaching the engine. The full optimization and solver logic of Maestro is not the focus of this dissertation. The relevant level of description is the engine's formal input contract: the structures it consumes, the outputs it produces, and the requirements that limits must satisfy before they can be executed. The engine is described through this interface, rather than through the details of its internal model-building or solving procedures.

3.2.2 Scores, Attributes and Limits

Maestro relies on four main input structures: a score table, customer attributes, stimulus attributes and limits. Table 3.1 provides an overview of these structures, which are then discussed in detail.

Table 3.1: Maestro input structures.

Input	**Contents**	**Role**
Score table	Customer–stimulus pairs, each with a numerical value	Defines the assignment space and drives the objective
Customer attributes	Columns describing each customer (e.g. segment, region)	Provide conditions on customers
Stimulus attributes	Columns describing each stimulus (e.g. category, type)	Provide conditions on stimuli
Limits	Business restrictions in the four-element structure	Encode the restrictions the engine must enforce

The score table records the customer–stimulus combinations and associates each combination with a numerical value representing affinity or expected relevance. These values define the objective function, which maximizes the total value of the selected assignments. The rows of the score table define the assignment space over which the engine can create decision variables. Each customer–stimulus pair represented in the table corresponds to a candidate assignment that may or may not be selected in the optimized solution.

The second and third input structures are the customer and stimulus attribute datasets. The customer dataset describes each customer through different attribute columns, such as profile, segment, region or other business-relevant characteristics. Likewise, the stimulus dataset uses columns such as category, type, product group or other properties relevant to the personalization product. These attributes are not optimized by the engine. Their role is to provide characteristics that can be used to define restrictions, for example to apply a rule to a customer segment, restrict a stimulus to a region, or cap the assignments within a product category.

The fourth input structure is the set of formal constraints, referred to as limits. A limit is the formal representation of a business restriction in a form that the engine can enforce. Three terms are used consistently throughout the dissertation to name the successive forms a rule takes, as summarized in Table 3.2.

Table 3.2: Terminology for the successive forms of a business rule.

Term	**Domain**	**What it denotes**
Restriction	Business	The rule as the user states it in natural language
Limit	Engine	Its structured form in the four-element representation the engine accepts
Constraint	Mathematical	The aggregate inequality the engine derives from the limit and imposes on the optimization

Each limit specifies four main elements: the aggregation level, the affected subset, the operator and the bound. The aggregation level indicates whether the restriction is evaluated per customer or per stimulus. The affected subset identifies the assignment variables to which the restriction applies, either through explicit customer or stimulus identifiers, or through conditions over customer attribute columns, stimulus attribute columns, or both. The operator defines whether the aggregated quantity is constrained by a maximum, a minimum or an equality condition. The bound defines the numerical value against which that quantity is compared. From these elements, Maestro builds the constraint for each aggregation unit. It sums the assignment variables in the affected subset and then compares that sum to the bound. The restriction “Each customer can receive at most 2 products”, for example, has a per-customer aggregation level and a maximum bound of 2. For each customer, Maestro sums that customer’s product assignments and requires the total to be at most 2.

These four elements can represent restrictions such as a maximum number of stimuli per customer, a minimum audience per stimulus, or a restriction limited to customers or stimuli with a

given attribute value. For a limit to be enforced, however, each element must be consistent with the prepared inputs and with the structure Maestro supports. The aggregation level must be one that the engine supports. The operator must be admissible. The bound must be a valid numerical value. The affected subset must reference customer or stimulus identifiers, attributes or values that exist in the prepared inputs, and any condition over those attributes must be expressed in a form the engine can process. A missing identifier, a missing attribute, an invalid value, an unsupported operator, an incorrectly structured subset condition, or an incompatible set of limits can prevent enforcement or make the optimization model infeasible. Correct limit specification is therefore what turns a business restriction into an enforceable constraint.

3.3 Current Limit-Specification Process

Business restrictions only become executable by Maestro once they are expressed as formal limits. This section examines how that transformation is currently performed. It first describes the process through which restrictions expressed in business language are translated into formal limits, and then identifies the main limitations and risks of that process. The analysis concerns limit specification, not the broader personalization operation.

3.3.1 As-Is Process

The data inputs required by Maestro are part of the structured preparation of personalization campaigns. They define the assignment space and provide the customer and stimulus attributes and values used to define restrictions. The scenario-specific input is the set of limits, through which each business restriction is represented in a form the engine can enforce.

Producing these limits is mediated by technical expertise. A business user states one or more restrictions in natural language, using the terms in which campaign rules are discussed. A technical specialist first interprets each rule and then, where possible, translates that interpretation into the four elements of a limit: aggregation level, affected subset, operator and bound. In the current process these two steps — interpreting the intent and formalizing it as a limit — are performed together, manually, by the same specialist. When the rule involves an affected subset, the technical work includes aligning the relevant attribute columns and values with their exact representation in the prepared customer or stimulus data, and organizing the condition into the affected-subset structure that Maestro uses to restrict the limit to the relevant assignments. Once specified, the limits are provided to Maestro together with the score and attribute inputs for execution. When execution produces an allocation, Maestro returns the optimized assignments, which serve as the allocation result. When the model is infeasible, or when the allocation does not reflect the original business intent, that outcome is reviewed, the limit specification is adjusted, and the cycle repeats. The limit-specification process is illustrated in Figure 3.2.

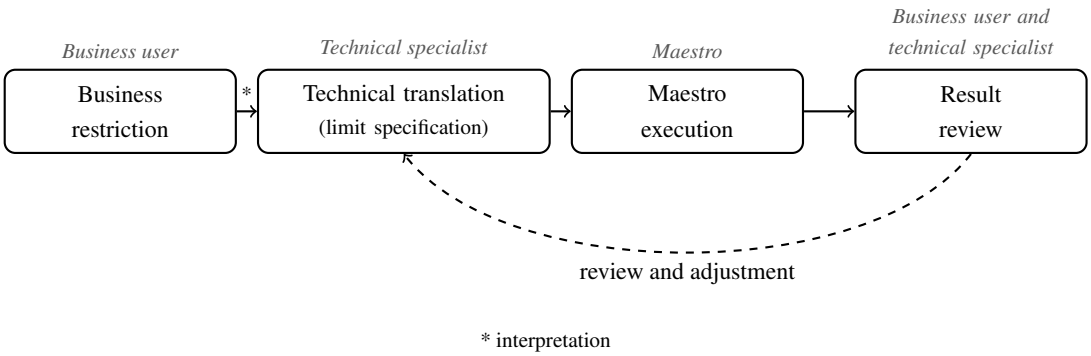


Figure 3.2: Limit-specification process.

The technical translation is not uniform: the effort required to specify a limit varies with the restriction. Some restrictions map almost directly onto the formal fields of a limit. In a personalized leaflet setting, the restriction "Each customer can receive at most 2 products" is represented with a per-customer aggregation level, a maximum operator and a bound of 2, without any affected-subset condition.

Restrictions that refer to subsets involve further specification work. In a personalized leaflet setting, the restriction "Customers from Madeira or the Azores can receive at most 1 hygiene product" is no longer defined only by an operator and a bound. Its formal representation includes an affected subset with two conditions: a regional condition over customers and a product-category condition over stimuli. The regional condition uses the relevant customer attribute and includes Madeira and the Azores as two accepted values. The product-category condition uses the relevant stimulus attribute and restricts the stimuli to the exact value used for hygiene products in the prepared data. Together, the regional and product-category conditions define the affected subset, so that the limit applies only to assignments satisfying both. When restrictions involve several attributes, multiple values within the same attribute, or conditions that do not correspond directly to a single attribute value, the specification becomes more demanding. The challenge is structural as well as numerical: the business condition must be translated into a representation that the engine can process.

3.3.2 Limitations and Risks

The current limit-specification process produces the limits required by Maestro. The limitation is not the absence of a formal mechanism, but the dependence on manual technical mediation to translate business restrictions into that mechanism.

Business users state the intent of a restriction, but they do not turn it into a Maestro-ready limit on their own. Producing such a limit requires familiarity with the engine's formal input contract: how aggregation levels, affected subsets, operators and bounds are structured, how input data attributes and values are referenced within that structure, and which conditions the engine can process. This dependency appears first when assessing whether a stated intent can be expressed as a valid restriction. A requirement may be natural in business terms but may have no direct or

acceptable formal expression, and recognizing this requires technical judgment. It appears again in the specification itself, where the restriction is represented through the four elements of a limit and through structures accepted by the engine. It appears once more after execution, when an infeasible result or a result that clearly does not reflect the original business intent has to be reviewed; the specification is then adjusted, and in some cases the intent itself has to be reconsidered, before the cycle can be repeated. At each of these points, progress depends on technical specialists whose availability and priorities may not coincide with the need for rapid scenario exploration. The path from intent to result becomes longer, and the autonomy of the users who best understand the intent is reduced.

This dependency also creates a risk that is more subtle than delay. The translation is performed by specialists, but it remains manual; meaning can still be lost or distorted. A restriction may be mapped to the wrong attribute, refer to a value that is not represented in the prepared data, use an unsupported operator, or be represented through an incorrectly structured affected subset. Some of these problems are visible because they prevent execution or make the model infeasible. Others are not: a limit can be well formed and formally valid while still encoding a meaning different from the one the business intended. In that case, Maestro optimizes the formal problem it receives, which may not correspond to the original restriction. A technically successful execution is therefore not, by itself, evidence that the original business intent was faithfully represented, giving operational form to the reliability concern established conceptually in the literature review. For business users, this mismatch is difficult to detect. They mainly see the optimized assignments, not the technical choices that produced the configured limit: which attributes and values were used, or how the limit relates to the original request. As restrictions become more numerous or complex, it becomes harder to assess whether the result reflects the intended business rule. This weakens confidence in the optimized decision and limits the practical value of the optimization-based process.

Taken together, these points describe a single, concrete issue. Business restrictions are expressed in one representation system, while Maestro requires another: structured, formal and compatible limits. The current bridge between the two is manual and dependent on technical expertise, which leaves it vulnerable to semantic error and difficult for business users to verify. The problem addressed in this dissertation is not the optimization engine itself, but the dependency on this manual bridge between business intent and Maestro-compatible formal limits, where correctness, compatibility and transparency are difficult to preserve.

Chapter 4

Methodology

The present chapter elaborates on the methodology proposed to address the dependency on a manual bridge between business intent and Maestro-compatible formal limits. It first establishes the methodological approach and the assumptions that delimit the proposed method. It then formalizes the limit structure that the method must produce, before describing how restrictions are grounded and represented explicitly. The chapter subsequently presents the design of the controlled semantic layer, including the architecture and the procedure by which a business restriction is interpreted, validated and assembled into a Maestro-compatible limit.

4.1 Methodological Approach

The methodology followed an incremental engineering logic centered on a single technical question: how a business restriction can be represented as a formal limit that Maestro can enforce. The approach began with an analysis of the current limit-specification process and of the engine's formal input contract, which fixed the target of the method. This analysis then enabled the definition of the structured intermediate representation required to connect natural-language restrictions and formal limits, and finally the specification of the deterministic control that confirms each interpretation and assembles it into a limit.

This logic determined the methodological sequence. The formal input contract of Maestro was analyzed first, identifying the limit structure as the integration target and the conditions it imposes on what a restriction can express. A controlled semantic layer was then designed to mediate between business users and the engine. The layer uses the inputs that the engine already requires and introduces a structured intermediate representation through which a natural-language restriction is made explicit before validation and assembly. Deterministic validation was specified as a mandatory step between semantic interpretation and execution, so that compatibility is confirmed before any restriction reaches the engine. The final stage prepared the design for implementation and validation. The sequence of the methodology was therefore determined by the need to produce limits that preserve an explicit link to the interpreted business intent while remaining compatible with the engine. The result of this sequence is the core method described in the remainder of the

chapter: its inputs, the structured representation it produces, and the separation of responsibilities through which a restriction is interpreted, validated and assembled into a limit.

4.2 Design Features and Assumptions

The proposed method rests on a set of design features and assumptions that fix the technical ground before the formalization. They are summarized as follows:

- Maestro is preserved as the formal optimization engine, and its core is neither redesigned nor replaced. The proposed solution operates upstream of the engine.
- The target of the method is the limits structure that Maestro accepts. A restriction can be integrated only if it can be expressed as an aggregate linear limit over the binary allocation variables: a sum of assignment variables, grouped by an aggregation level and compared with a bound through a relational operator.
- The assignment space is defined externally. The customer, stimulus and score inputs are produced by the structured preparation of each campaign, upstream of the method. The engine creates a decision variable only for the customer–stimulus pairs present in the score table, and the method operates over this prepared set of eligible pairs.
- The affected subset of a restriction is defined by matching customer or stimulus attributes to explicit values. Customer attributes are defined per customer and stimulus attributes per stimulus. Attributes that depend jointly on the customer–stimulus pair, such as an affinity score defined for each specific pair, fall outside the core method, since they cannot serve as an independent customer or stimulus condition.
- The LLM interprets the restriction and maps it to a candidate structured representation. It does not validate data or execute the optimization.
- The domain description constrains interpretation by exposing the available attributes and example values that can be referenced. It may not list every possible value, so suggested references are confirmed only when matched against the prepared customer and stimulus data the engine consumes for the use case.
- Deterministic validation confirms that the candidate is compatible with the prepared data and the engine's input contract before assembly.

These assumptions delimit the core method. It addresses restrictions whose four elements — the aggregation level, the affected subset, the operator and the bound — can each be filled with a fixed and explicit value taken directly from the restriction. A restriction falls outside the method when any of these elements cannot be resolved in that direct way. This happens in three cases. The aggregation level may not be one that the engine supports, as in a global restriction that caps the total number of assignments across all customers and stimuli at once, rather than

per customer or per stimulus. The affected subset may not reduce to attributes matched against explicit values, as in a condition that selects customers by a threshold, such as those older than a given age, rather than by a stored value. Such a condition could only enter the method if the threshold were pre-computed into a stored attribute beforehand, which is itself a data-preparation step outside the method. Or the bound may not be a fixed number but a quantity that depends on the data or on other assignments, as in percentage-based conditions or in a restriction that ties one group's allocation to another's. In all these cases, the restriction cannot be carried directly into a limit and is therefore not handled by the core method.

4.3 Formal Limit Structure

Business restrictions must be represented in a structure that the engine can execute. The controlled semantic layer does not formulate a new optimization model; Maestro already defines one. The purpose is to formalize the structure that the layer must produce so that a restriction becomes enforceable. Table 4.1 summarizes the notation used throughout this formalization.

Table 4.1: Notation used in the restriction formalization.

Symbol / term	Definition
Sets	
C	Set of customers, indexed by c
S	Set of stimuli, indexed by s
$E \subseteq C \times S$	Set of eligible customer–stimulus pairs defined by the score table
Limit representation	
$\ell = (g, Q, \theta, b)$	Maestro-compatible limit
g	Aggregation level
$\theta \in \{\leq, \geq, =\}$	Relational operator
b	Bound
Derived subsets	
$Q \subseteq E$	Affected subset of a restriction
$U_{g,Q}$	Aggregation units induced by g after applying affected subset Q, indexed by u
Q_u	Eligible pairs in Q associated with aggregation unit u
Attributes and conditions	
K	Set of grounded conditions defining Q, indexed by k
p_k	Grounded predicate over an eligible pair associated with condition k
a_k^C	Customer-side attribute function associated with condition k
a_k^S	Stimulus-side attribute function associated with condition k
V_k	Set of admissible values for condition k
Decision variable	
x_{cs}	Binary assignment variable for eligible pair $(c,s) \in E$

Personalization is modeled as a customer–stimulus allocation. Let C be the set of customers

and S the set of stimuli. Not every pair is admissible: the score table defines the eligible customer–stimulus pairs over which the engine creates decision variables. Each eligible pair corresponds to a candidate assignment. The set of eligible customer–stimulus pairs is denoted by E and defined in Equation (4.1).

$$E \subseteq C \times S \tag{4.1}$$

For each eligible pair, a binary decision variable expresses whether a stimulus is assigned to a customer, as defined in Equation (4.2).

$$x_{cs} \in \{0,1\}, \qquad (c,s) \in E, \tag{4.2}$$

The variable x_{cs} takes value 1 if stimulus s is assigned to customer c, and 0 otherwise. Each score is the coefficient that the corresponding variable x_{cs} contributes to the objective, while restrictions limit the assignments that may be selected together. The decision unit is always an eligible customer–stimulus pair, and the restrictions targeted by the semantic layer are expressed as aggregate expressions over these variables.

A formal limit specifies how such an aggregate restriction is represented for execution by the engine. The content of a restriction enforced by Maestro can be described as a tuple, defined in Equation (4.3).

$$\ell = (g, Q, \theta, b), \tag{4.3}$$

where g is the aggregation level, Q is the affected subset, θ is the relational operator, and b is the bound. The operator takes the relational form associated with the maximum, minimum and equality limit types, as given in Equation (4.4).

$$\theta \in \{\leq, \geq, =\}. \tag{4.4}$$

The limit terminology is formalized here and mapped to the engine's input contract. The aggregation level g determines where the limit is stored among the engine's per-customer and per-stimulus limits, while the affected subset Q, the operator θ and the bound b populate, respectively, the subset, operator and bound components of that limit.

The affected subset Q determines the scope of the limit. It contains the eligible assignments to which the restriction applies and is defined by conditions over customer or stimulus attributes. Each condition $k \in K$ is represented as a grounded predicate p_k over an eligible pair. A predicate is induced either by a customer attribute, such as $a_k^C(c) \in V_k$, or by a stimulus attribute, such as $a_k^S(s) \in V_k$. The affected subset contains the eligible pairs that satisfy all conditions, as defined in Equation (4.5).

$$Q = \{\, (c,s) \in E : p_k(c,s) = 1,\ \forall k \in K \,\}. \tag{4.5}$$

Within a single condition, several values combine as a disjunction, so that any value in V_k qualifies. Across conditions, the requirements combine as a conjunction, so that a pair must satisfy all of them. When no condition is given, $Q = E$ and the restriction applies to all eligible assignments. Because attributes are defined per customer or per stimulus, a condition is always grounded on one side of the assignment, and a subset may combine customer-side and stimulus-side conditions.

The aggregation level determines the units over which the assignments are summed. Let $U_{g,Q}$ denote the aggregation units induced by g after applying the affected subset Q. When no affected subset is specified, $U_{g,Q}$ corresponds to all units of the aggregation level. Let Q_u denote the pairs of the affected subset associated with unit u. In this formalization, constraints are induced only for aggregation units with at least one qualifying pair in Q. Thus $U_{g,Q}$ is the set of units for which $Q_u \neq \emptyset$. A unit with no qualifying pair is left unconstrained, since imposing a limit on a unit with no eligible assignment to satisfy it would be trivially infeasible. This matters in particular for a minimum, which such a unit could never meet. For a customer-level limit, Q_u contains the qualifying assignments of customer u; for a stimulus-level limit, it contains the qualifying assignments of stimulus u. For each aggregation unit, the limit induces an aggregate constraint. The sum of the assignment variables in that unit is compared to the bound b through the relational operator θ, as written in Equation (4.6).

$$\sum_{(c,s)\in Q_u} x_{cs} \;\theta\; b, \qquad \forall u \in U_{g,Q}, \quad \theta \in \{\leq, \geq, =\}. \tag{4.6}$$

Here θ is the relational operator, not a multiplier. Each choice of θ instantiates one of the three limit types — maximum ($\leq$), minimum ($\geq$) or equality ($=$). This schema captures the core count-based limit structure targeted by the proposed layer. It does not redefine the optimization model, which the engine builds internally from each limit.

The restriction "Customers from Madeira or the Azores can receive at most 1 hygiene product" illustrates the structure. The aggregation level g is per customer. The affected subset Q is defined by two conditions: a customer-side condition on the region attribute, with the admissible values Madeira and Azores combined as a disjunction, and a stimulus-side condition on the product-category attribute restricted to hygiene products. The two conditions combine as a conjunction. The operator θ is a maximum, expressed as $\leq$, and the bound b is 1. The aggregation units $U_{g,Q}$ are then the customers covered by the affected subset, and for each such customer u the limit imposes $\sum_{(c,s)\in Q_u} x_{cs} \leq 1$ on that customer's assignments of hygiene-product stimuli. Additional illustrative examples of restriction formalization are provided in Table A.1 of Appendix A.

4.4 Grounding and Intermediate Representation

A natural-language restriction must be grounded and made explicit before it can become a limit. Grounding anchors interpretation in the target domain, and the intermediate representation records that interpretation in a form that can be inspected and validated. Both rely on the domain description.

The domain description provides the controlled context for interpretation. For a given use case, it is built from the prepared data, exposing the available customer and stimulus attributes, their data types, and example or admissible values in their expected format. It may also include manually provided explanatory notes for attributes whose meaning is not self-evident. Grounding interpretation in this description means that a business term is mapped to a specific attribute and to admissible values within that attribute, rather than to an open reading of language. This follows the principle that interpretation must be anchored in an explicit description of the domain rather than in language alone (Quamar et al., 2022).

The domain description is not only context for the LLM; it also defines what can be referenced in the candidate representation. It supports interpretation by exposing what can be referenced, but it does not itself confirm that a candidate is valid. An attribute that is absent from the description cannot be used for grounding. A referenced value that is not listed, however, is not invalid for that reason alone, since a description that records example values may not contain the full set of values in the prepared data. Such a value must be checked against the prepared data before it can be accepted. The domain description therefore grounds interpretation; it does not replace the prepared data. Because the description is specific to each use case, the same core method can extend to additional configured use cases by changing the use-case-specific inputs rather than the core procedure.

The LLM does not produce an executable limit directly. It produces a candidate structured representation $\hat{z}$: an explicit intermediate representation that records the interpreted restriction in fields aligned with the limit structure. The representation captures the aggregation level, the operator and the bound, together with the customer-side and stimulus-side conditions that define the affected subset and the grounded attributes and values they reference. These fields correspond to the elements of the limit tuple $\ell = (g, Q, \theta, b)$ introduced in Equation (4.3): the aggregation level, the affected subset, the operator and the bound.

The representation is intermediate by design. It is explicit enough to be inspected and validated, but it is not yet a formal input for the engine. Its purpose is to keep semantic interpretation separate from the deterministic validation and assembly stages that follow, providing a stable structure on which those stages can operate.

4.5 Controlled Semantic Layer Design

The procedure by which a natural-language restriction becomes a Maestro-compatible limit follows the staged sequence shown in Equation (4.7).

$$(r, D) \xrightarrow{\text{LLM}} \hat{z} \xrightarrow{V} z \xrightarrow{T} \ell \rightarrow \text{Maestro}. \tag{4.7}$$

In this sequence, r denotes the natural-language restriction and D the domain description, supplied together to the LLM. The remaining symbols denote the candidate structured representation $\hat{z}$, the deterministic validation procedure V, the validated representation z, the assembly step T,

and the Maestro-compatible limit ℓ. A restriction advances only if it satisfies the requirements of the current stage; otherwise, the system returns specific feedback explaining the issue.

The transition from r to $\hat{z}$ corresponds to semantic interpretation. The orchestration supplies the restriction and the domain description to the LLM. The LLM uses this context to produce $\hat{z}$, making the interpretation explicit in structured form. The proposed representation is treated as a semantic hypothesis rather than as a final constraint, consistent with the reliability limitations of LLM outputs (Ji et al., 2023).

The validation procedure V applies deterministic checks to the candidate representation before it can be assembled. These checks are organized in two complementary steps. Attribute and value checks come first, confirming that the customer and stimulus references proposed in $\hat{z}$ can be matched against the prepared data, since a representation that names a value or attribute absent from the data cannot yield a valid limit regardless of its structure. Limit-structure checks then verify that the four elements of the limit are compatible with the engine's expected format: the aggregation level must be supported, the affected subset must be structured in a form the engine can read, the operator must be admissible, and the bound must be a valid numerical value. Validation prevents structurally invalid, unsupported or data-incompatible candidates from reaching execution, following the principle of rejecting invalid candidates before execution (Scholak et al., 2021). It does not guarantee that the full set of restrictions is jointly feasible, since feasibility is determined only during optimization.

The assembly step T places the validated representation in the limits structure consumed by the engine. It does not optimize and does not reinterpret the restriction: each element of the validated representation is placed in its corresponding field of the limit, and the aggregation level determines where the limit is stored.

The staged sequence specifies the procedural flow; the resulting architecture is illustrated in Figure 4.1.

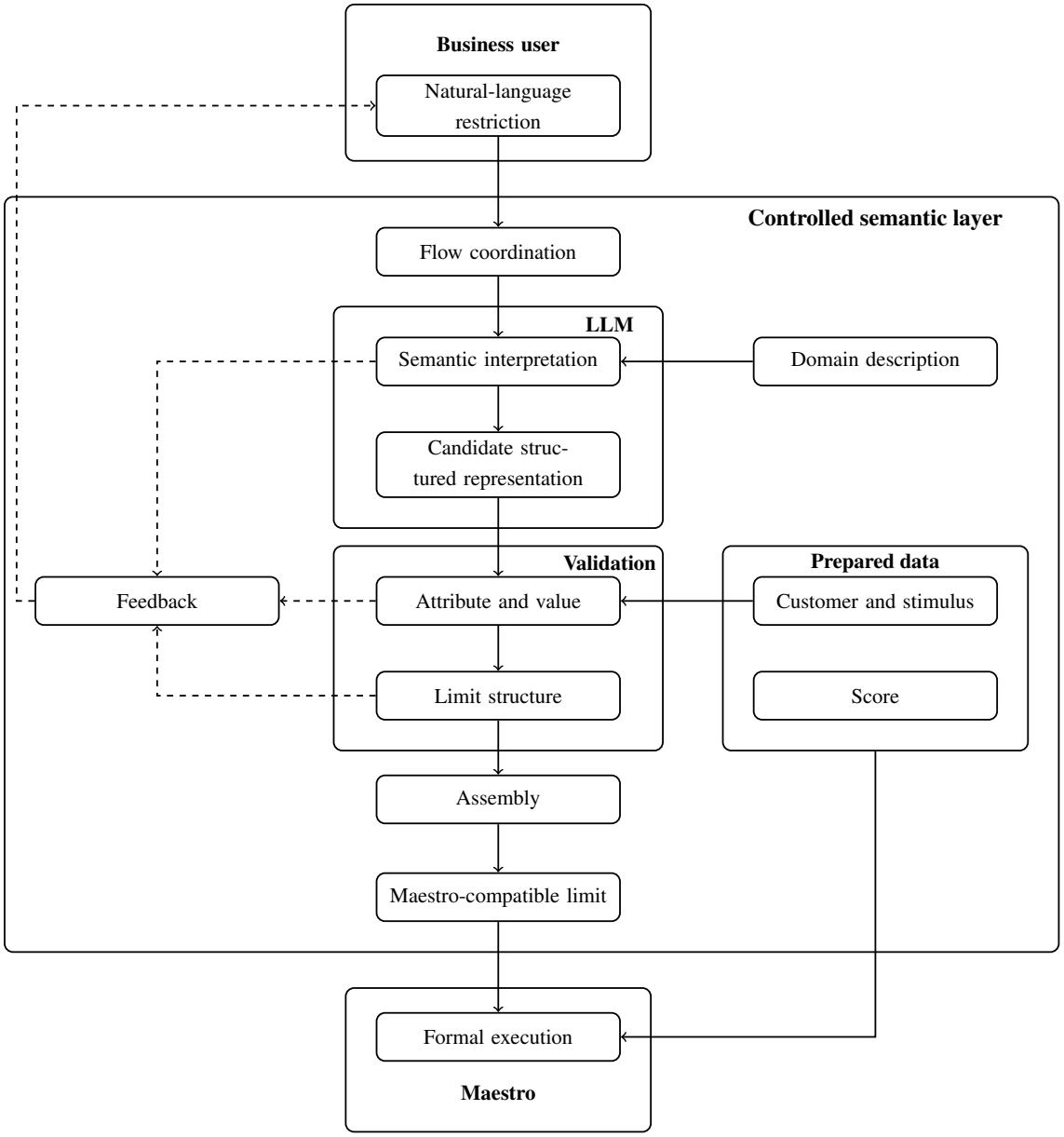


Figure 4.1: Conceptual architecture of the controlled semantic layer.

The controlled semantic layer therefore defines a constrained bridge from business intent to Maestro-compatible limits. Its contribution is not to replace Maestro or solve the allocation problem, but to make restriction specification explicit, inspectable and controlled. The LLM proposes a structured interpretation, deterministic validation determines whether that interpretation can become a valid limit, and formal execution remains reserved for Maestro. The method addresses the manual translation dependency while preserving the engine's formal role.

Chapter 5

Implementation and Results

This chapter presents the development of the solution and the progressive results obtained throughout its implementation and validation. Section 5.1 describes the implementation and validation of the proposed controlled semantic layer, establishing the central architecture of this dissertation as a working MVP that proves the proposed concept end-to-end. Section 5.2 then presents a set of extensions that, while outside the initial problem scope, adapt the validated solution to the conditions of real business use. Finally, Section 5.3 discusses the validation evidence as a whole and the nature of what it establishes. Across the chapter, the results reported are the controlled solution itself and its progressive extensions. They are demonstrated to address the gap between business language and the formal limits the engine requires, and shown to operate under conditions representative of real business use. The evidence is technical and functional, drawn from realistic restrictions over prepared data rather than synthetic inputs, covering the scenarios such use produces. Quantifying the operational benefit this enables falls outside the scope of this work.

5.1 End-to-End Implementation

The first development cycle implemented and validated the proposed architecture as an executable workflow, connecting natural-language input, formal limit generation, optimization execution and result feedback. The section first defines the initial development setting and then describes the pipeline and its main components. It then presents the validation of the implemented workflow across representative input scenarios, and the refinement introduced to make repeated execution practical.

5.1.1 Initial Setting

The MVP was developed under a deliberately constrained initial setting, to assess the base workflow. It was limited to one use case, one restriction per interaction, and a controlled subset of the prepared real data. The selected use case corresponded to the weekly personalized leaflets. Each interaction considered a single business restriction, allowing the behavior of the workflow

to be analyzed without the additional complexity of compound requests. The prepared data subset provided the necessary domain information for interpretation, validation, and execution, while keeping the development environment stable and observable. The solution operates in Portuguese, the language in which business users state restrictions and receive feedback. The example restrictions shown throughout this chapter are presented in English for readability.

This configuration supported the assessment of the base workflow before introducing broader input capabilities, richer expressiveness, operational configuration and larger data volumes. The result was a validated base workflow, with clearly defined implementation conditions and a foundation for the developments that follow.

5.1.2 Implemented Pipeline

The proposed architecture established the components of the controlled semantic layer and the order in which a restriction moves through them, as set out in Figure 4.1. The development followed this sequence directly: each stage was built as an independent module, and the pipeline was assembled along the established flow. Each module contains its own logic and functions, while the overall workflow is coordinated by a central orchestration layer. Use-case-specific configuration is kept separate from the core processing logic. The components are presented in the order in which a restriction traverses the pipeline.

5.1.2.1 User Interface

The User Interface (UI) is the entry point through which the business user writes a restriction in natural language and receives the corresponding outcome. It was implemented as a notebook within the same data and analytics platform as the rest of the solution, rather than as a separate application. It is designed to keep the interaction simple and focused, with only the guidance needed to formulate a clear business restriction.

The interface provides short instructions without imposing a formal syntax. The guidance includes example restrictions, such as “Each customer can receive at most 3 products”, “Each product must be assigned to at least 100 customers”, and “Customers from Madeira can receive at most 1 product”. It also indicates the basic writing rules: each input should contain one restriction, include a numeric limit, and may include filters such as region, customer segment, or product category.

This design keeps the interaction close to business language while giving the user a minimal structure for writing restrictions. The UI does not require technical knowledge of the optimization model or of the formal limit structure used by Maestro. Its role is to provide a clear and simple point of interaction between the user and the system, while the processing of the restriction is handled by the solution itself.

This initial interface supported the single-restriction workflow. It was extended alongside the capabilities described in Section 5.2, and is shown in its matured form in Appendix G.

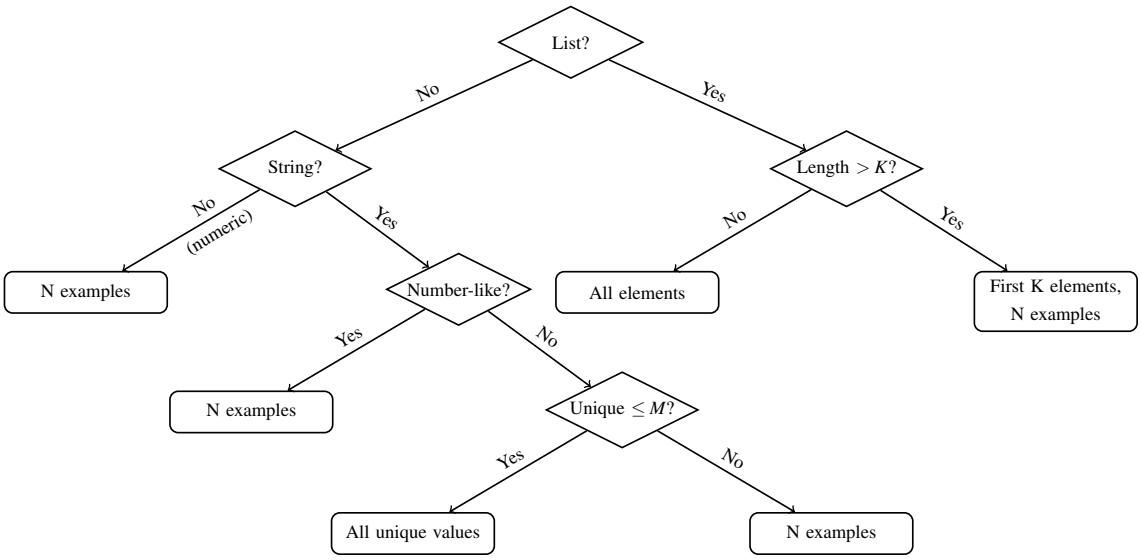
5.1.2.2 Data Preparation and Domain Description

Two data modules were developed to support the use of prepared data within the workflow. The first module handles access to the data, while the second builds the domain description used by the LLM during interpretation.

The first module defines the logic to read the prepared dataset, apply optional volume filters, and align the customer, stimulus and score fields with the names expected by the pipeline. It imports and prepares the data over which restrictions are interpreted, validated and executed.

The second module builds the domain description. One approach would provide the LLM with the complete prepared data, which in principle would let it resolve any restriction, including those requiring a computation such as a threshold or a count, by performing the calculation directly over the data. This is not viable. The prepared data reaches billions of rows, far beyond what fits in the LLM's context, so it cannot be placed in the input regardless of cost. The domain description is the viable alternative. It provides a structured summary of the available attributes, their types, whether they contain single values or lists, and examples of values in the expected format, enough to ground interpretation without the full data. Figure 5.1 summarizes the rule-based logic used to decide how examples are included for each attribute type.

A direct consequence of summarizing the data rather than reproducing it is that some restrictions cannot be expressed through the domain description. Which restrictions these are, and why each falls outside what a limit can represent, is detailed in Section 5.1.2.3, when the interpretation step is described.



In this setting: N = 15 example values, M = 100 distinct-value threshold, K = 5 list elements. These thresholds are configuration parameters; the values shown are those of the present use case.

Figure 5.1: Example-selection logic used in the domain description.

The amount of information included in the examples field varies according to the attribute type, governed by the thresholds shown in Figure 5.1. Numeric attributes receive a limited number of examples, since interpretation depends primarily on identifying the correct attribute. For example, in a restriction such as "Customers with 3 children can receive at most 5 products", the LLM only needs enough information to associate the concept of children with the corresponding attribute. The specific value may not appear in the examples and can still be interpreted correctly.

Categorical attributes receive richer representation because interpretation depends on matching business terms to existing values in the prepared data. Consider a restriction such as "Products from the sweets category must be assigned to at least 100 customers". The LLM must identify both the relevant attribute and the corresponding value. If the domain description does not expose the relevant category values, the LLM may generate a plausible but non-matching value, such as "Sweets" instead of the stored value "Candy". Low-cardinality categorical attributes, those with at most M distinct values, expose all of them, while high-cardinality text attributes expose N representative examples.

This construction gives more information to attributes where interpretation depends on specific values and less information to attributes where type and format are sufficient. This reduces unnecessary context, limiting input size and noise while preserving the information most relevant for grounding. Further rules handle specific formats, such as attributes that hold codes or identifiers and attributes whose values are lists. These are detailed in Appendix B, together with the effect of this type-based selection on input size compared with an unselective description, reported in Table B.1.

5.1.2.3 Semantic Interpretation and Structuring

Two modules implement the stage in which a natural-language restriction becomes a candidate structured representation: the connection to the LLM and the prompt that governs its behavior. The connection module contacts the LLM through an Application Programming Interface (API) and exposes a single call used by the rest of the pipeline. GPT-4.1-mini was used as the initial LLM during development, providing a baseline for the implementation and validation of the workflow.

The prompt is the central component of this stage, as it constrains the LLM to act as a semantic interpreter that produces the candidate structured representation. It was designed around what the LLM must accomplish when it receives a restriction, and it is described here by following that sequence. The LLM operates with a fixed context. The problem is an assignment between customers and stimuli, and the only admissible output, in this initial single-restriction setting, is a single structured representation expressed through a dimension, an operator, a numeric bound, and an optional affected subset. The LLM also receives the domain description in the form defined earlier. On this basis, it performs two reasoning steps on each restriction: it decides whether the restriction can be expressed and, when it can, produces the structured representation.

The first step determines whether the restriction can be represented directly as a limit. The criterion is structural rather than a list of accepted phrasings: a restriction is rejected when the information required to form a limit is absent, or when expressing it would require a transformation

that the limit structure does not support, as anticipated in Section 5.1.2.2. Any restriction that cannot be placed directly into the formal limit structure is rejected at this step with a specific error in the user's language, rather than approximated. The LLM does not compute values, infer bounds from the data, or reshape the output. Table 5.1 illustrates representative categories rejected at this step, grouped by whether the required information is absent or the limit structure cannot express it, each with a representative case.

Table 5.1: Representative restriction categories rejected at interpretation.

Rejected category	**Representative restriction**
No per-entity dimension (global)	"Exactly 500 products must be assigned."
No explicit numeric bound	"Each customer should receive a few products."
Optimization objective, not a constraint	"Maximize the number of assigned products."
Comparison or threshold on a value	"Customers with value above 3 children can receive at most 3 products."
Percentage or relative limit	"Each customer must receive half of the available products."
Count of elements within a value	"Products whose category list has more than two entries must be assigned to at least 100 customers."
Limit or filter dependent on dataset content	"Madeira customers receive twice the products of Azores customers."
No attribute represents the intent	A restriction over a concept absent from the prepared data

The structured form of this validity criterion, and the way these distinctions are specified to the LLM, are presented in a structured prompt excerpt in Figure C.1 of Appendix C.

When a restriction is admissible, the LLM produces the structured representation. The dimension, operator and bound follow directly from the intent — whether assignments are counted per customer or per stimulus, whether the limit is a maximum, a minimum or an equality, and what the numeric value is. The non-trivial part is the affected subset, when the restriction filters by an attribute. The prompt specifies a precise resolution procedure, which determines whether the later exact match against the prepared data succeeds.

The procedure resolves an attribute reference in phases. The LLM first determines whether the intent corresponds to an available attribute, selecting the one that most directly represents the concept and never introducing an attribute absent from the domain description; if no attribute represents the intent, the restriction is rejected. Once an attribute is selected, the LLM resolves the value. If the user's term is present among the values shown for that attribute, that stored value is used. If it is not present literally but matches one by meaning — a different wording, a singular or plural form, a minor variation — the stored value is still used. For example, a restriction limiting "Biologic products" resolves to the stored value "Organic" for the product-type attribute, because the description exposes that value. Only when no value matches does the LLM use the user's own term, normalized to the formatting conventions observed in that attribute's own values

rather than copied *verbatim*. The normalization is performed per attribute, on the evidence of that attribute's values, and adjusts casing, prefixes, accents and, for structured values, element ordering, only where the values give clear evidence for the change. For example, a restriction referring to "ração" for a product-type attribute whose stored values are capitalized and unaccented resolves to "Racao", because the attribute's own values provide that evidence.

This handling is necessary because the solution operates in Portuguese, where accents and similar variations are frequent and a term written by the user often differs in form from the stored value, so the LLM must reconcile the two without altering the underlying value. The fallback exists because the domain description is not guaranteed to contain every value, and its purpose is to give the produced value the best chance of matching the prepared data exactly. This is also where the design of the domain description pays off: the richer the values exposed for an attribute, the more often a restriction resolves by direct or semantic match and the less the LLM relies on normalization.

Finally, the LLM produces the value in the structure of the attribute, because the subsequent validation is an exact match and a structurally divergent value would fail it even when the intent is correct. The output value must follow the attribute's data type and structural shape — a single value, a list, or a nested list — without flattening or adding nesting. This structure governs both the produced format and the interpretation of the restriction. This can be seen by comparing two regional attributes. A customer-region attribute holds individual values, such as "Madeira" or "Azores" separately, whereas a product-region attribute holds lists, where "[Madeira, Azores]" is itself a single stored value, distinct from "Madeira" and "Azores" individually. The restriction "Customers from Madeira and the Azores can receive at most 1 product" therefore resolves to a customer-region condition with the two independent values "Madeira" and "Azores". Because each customer belongs to a single region, the two values act as alternatives, and the condition covers every customer whose region is Madeira or Azores. The restriction "Products from Madeira and the Azores must be assigned to at least 100 customers" behaves differently, resolving to the single product-region value "[Madeira, Azores]", a stored list distinct from "Madeira" or "Azores" alone. Distinguishing these cases is essential both for producing the value in the correct structure and for interpreting the restriction itself, and it directly affects how the subsequent match against the prepared data is performed.

Throughout, the LLM is instructed that it is not responsible for confirming whether a produced value exists in the prepared data; that confirmation is performed by the deterministic validation that follows. Its task is to interpret the restriction, decide whether it is expressible, and, when it is, produce a well-formed candidate representation in the expected structure.

5.1.2.4 Deterministic Validation and Execution

Before the candidate structured representation can be assembled into a limit and executed, it is submitted to deterministic validation. This validation applies the two complementary checks implemented as two distinct modules — one preceding the assembly, the other on the side of the engine — and the execution follows immediately once both checks succeed.

The first check confirms that the attributes and values proposed in the candidate representation correspond exactly to the prepared data. This is what distinguishes a candidate interpretation from a validated representation: an interpretation that is well-formed but references a value or an attribute absent from the data is rejected at this point, before execution. The check returns distinct outcomes: a validated success, which allows the restriction to advance, a no-correspondence outcome when a referenced value or attribute is not found, or an internal validation error. When the restriction cannot advance, the outcome drives specific feedback to the user, identifying the reason rather than returning a silent or generic failure.

The second check confirms that the four elements of the limit are compatible with the format the engine expects: the aggregation level must be supported, the affected subset must be structured in a form the engine can read, the operator must be admissible, and the bound must be a valid numerical value. It also applies validation rules to representations that are structurally well-formed but rely on information the engine cannot use as a condition. For instance, a restriction such as "Customers with an affinity score of 1 should receive at most 3 products" is rejected because the affinity score is not a customer attribute or a stimulus attribute. It describes a customer–stimulus relationship, since each value depends on a specific customer–stimulus pair, and therefore cannot serve as an independent customer or stimulus condition. Only representations that pass both checks are assembled into a Maestro-compatible limit.

Execution follows directly. The engine's own logic was incorporated through its existing implementation and left unchanged. The surrounding work was limited to building the customer, stimulus and score inputs it expects, assembling the converted limit, and running the optimization. The engine returns the status of the run — optimal, feasible but not proven optimal, infeasible, or error — together with the resulting allocations when a solution is produced and the technical output files generated by the solver. These outputs are written to a dedicated location for retrieval. The feasibility of the restrictions is determined only at this point, which is why it is distinct from the validation that precedes it: a representation can be valid and structurally compatible yet still be infeasible against the assignment space, since feasibility depends on the restrictions taken together rather than on any one of them in isolation.

5.1.2.5 Use-Case Configuration and Orchestration

All the modules above define logic only. They are independent, do not run on their own, and contain no information specific to a restriction or a use case. The information that is specific to a use case — where the prepared data is read from, how its fields map to the pipeline's expected fields, any optional manual attribute descriptions, which attributes belong to the customer and which to the stimulus, and the thresholds that govern how many example values the domain description includes — is isolated in a separate configuration module. Adding a use case is therefore a matter of providing such a configuration, without altering the shared logic, an arrangement that later enabled both larger-scale operation and reuse across use cases.

The orchestration logic coordinates the whole path, shown as flow coordination in Figure 4.1. Although the figure places it at the start of the flow, it is not a single step but the logic that drives

every transition, present at each stage rather than only at the beginning. When a restriction is submitted through the interface, it loads the logic modules and the relevant use-case configuration, prepares the data and the domain description, builds the prompt, calls the LLM, passes the result through the deterministic validation, assembles the validated representation into a limit, runs the optimization, and writes the outcome back to the interface. It is also the point at which feedback is enforced: at each stage that can reject an input, the orchestration logic detects the outcome, stops the flow, and returns a specific message to the interface.

5.1.3 Validation

A structured validation set was used to evaluate the behavior of the solution across representative categories of restrictions. It confirmed whether supported restrictions followed the expected interpretation, validation and execution flow, and whether unsupported or unmatched restrictions were stopped at the appropriate stage. The validation set was applied to the complete workflow rather than to the LLM in isolation, since the objective was to assess the behavior of the implemented solution as a whole.

Each case was evaluated through the same workflow it would follow in normal operation, and the stage at which a case was resolved is part of what the validation confirms: a supported restriction should reach execution, while an unsupported or unmatched one should be stopped at the stage responsible for detecting it. This reflects the guiding principle that the LLM does not need to interpret every input perfectly, provided that the deterministic stages prevent invalid restrictions from being executed.

The validation set comprises 75 cases organized into nine categories, summarized in Table 5.2. The categories follow a deliberate progression. The first block (A–D) confirms that the fundamental operations work correctly: that the workflow produces the right structure for each combination of dimension and limit type, applies filters to the correct attributes, resolves natural-language terms to stored values, and rejects what cannot be expressed as a limit. The second block (E–H) stresses the workflow with variations it must tolerate in real use: values the LLM has not seen in the domain description examples, severely misspelled input, filters combining multiple conditions, and phrasings that differ from the canonical form while expressing the same intent. The final category (I) tests the structural limits of the representation, where the data contains lists, nested lists, and structured fields that demand precise format handling from the LLM. Throughout, the filters, attributes and values used in the cases are those of the prepared data for this use case, so that the validation exercises real domain content rather than synthetic inputs.

Table 5.2: Validation test categories.

Category	Cases	What it validates
A	6	Combinatorial coverage of dimension and limit type
B	8	Filters over real attributes
C	5	Semantic mapping of business terms to stored values
D	9	Rejection of unsupported requests
E	9	Edge cases: out-of-sample values, zero limits, identifiers
F	4	Orthographic robustness
G	3	Composite filters within and across attributes
H	8	Alternative formulations
I	23	Advanced cases: list-valued and structured attributes

The choice of model for the interpretation step was treated as an empirical decision with a single variable: holding the prompt, the domain description and the validation set constant, candidate models were run on the same 75 cases, so that the LLM was the only difference between runs. The implementation had used GPT-4.1-mini as an initial choice made without a detailed model evaluation; with the validation set now formalized, this decision was revisited systematically. The candidates were selected on a clear rationale. Models specialized for unrelated modalities or for deep reasoning were excluded as functionally irrelevant. Among the remaining models, those more economical than the one initially in use were tested first, to determine whether any matched the requirement at lower cost, with the more capable and more expensive models retained as alternatives should none suffice. Cost was assessed on input, since each call carries the domain description and the prompt as input while the structured output is small, so the cost of an interaction is dominated by input size. Table 5.3 reports the comparison, ordered from the least to the most expensive model. Costs refer to the provider's listed input prices at the time of evaluation.

Table 5.3: Model comparison on the validation set.

Model	Input cost per 1M tokens	Cases as expected	Primary failure nature
GPT-5-nano	\$0.05	43/75	Structural: frequent invalid output, higher latency
GPT-4o-mini	\$0.15	55/75	Incomplete handling: dropped filter components, confused similar attributes
GPT-5-mini	\$0.25	73/75	Residual advanced edge cases
GPT-4.1-mini	\$0.40	65/75	Structural: dimension confusion and flattened nested structures
GPT-5.1	\$1.25	66/75	Over-expansion: multiple values where one is expected

Two of the candidates were not adequate for the interpretation task, failing in ways that made

their output unreliable. Among the three that were adequate, what set them apart was performance on the structurally complex restrictions of category I, not on the common ones. Cost did not predict capability on this task, as the most expensive candidate did not score highest. Model selection therefore considered not only aggregate accuracy and input cost, but also consistency in the core categories required for reliable execution, particularly the ability to produce valid structured candidates for supported restrictions and to reject unsupported requests rather than generating executable-looking outputs. On this basis, GPT-5-mini was selected as the most capable of the adequate candidates at the lowest cost, more capable than the LLM initially in use while being less costly. The detailed analysis of the comparison, together with the breakdown of results by category and model, is given in Appendix D.

With the selected LLM, 73 of the 75 cases matched the expected output on this run, and the LLM has produced all 75 in other observed runs. In the two deviations reported here, the candidate output was intercepted by the deterministic checks before reaching execution, so no observed run produced an executed restriction that violated the intended behavior. The distribution of the cases across the workflow stages is reported in Table F.1 of Appendix F.

These results support a bounded conclusion. The 75 cases are representative but not exhaustive, and the interpretation step carries natural variability across runs. The safety of the workflow therefore rests not on perfect interpretation by the LLM, but on the deterministic boundaries that prevent invalid candidates from being executed.

5.1.4 Execution Performance

With the end-to-end workflow implemented, the runtime of a complete interaction became the main concern for usable execution. Each interaction in the initial implementation required approximately 480 to 500 seconds. Of this, the interpretation, validation and execution steps accounted for only 30 to 60 seconds. The remaining time corresponded to preparation work independent of the submitted restriction, principally constructing the domain description by scanning the prepared dataset to extract its attributes, their types and example values. At the data volumes involved, this scan dominates the runtime of a first interaction. This step produces the same output for a given use-case configuration regardless of the restriction submitted.

The preparation was separated from per-interaction execution and constructed once per use case. Interactions then consisted only of interpretation, validation and execution. Table 5.4 reports the runtime effect of this separation on the controlled subset of the prepared real data used in this phase.

Table 5.4: Runtime effect of separating setup from per-interaction execution.

Execution mode	Preparation	Approx. total time
Before separation	Domain description rebuilt every interaction	∼480–500 s
After separation	Preparation constructed once and reused across interactions	∼30–60 s

The controlled path itself did not change; only the handling of preparation steps was modified. This enabled reuse of the workflow. The reported values are internal runtime measurements, used as technical evidence that repeated interaction became practical. They are not a measure of business benefit.

5.2 Extensions for Realistic Use

The controlled semantic layer was established as a validated and usable solution for the specific problem addressed in this dissertation. In relation to the conditions of real business use, however, it remains an MVP: while it demonstrates the central concept and architecture, several capabilities required for practical use in the company under study were only addressed once the core was established. This section presents the extensions that address them. These extensions go beyond the original problem boundary, and their purpose is to adapt the validated solution to more realistic usage conditions rather than to extend the concept itself. Each extension follows the same approach used in the first implementation: a capability needed for realistic use is identified, implemented, and exercised under conditions representative of practical use in the company under study, while the separation of responsibilities is preserved. The extensions are presented as a progression from strengthening the internal robustness of the solution on the input side, to operational configuration, scalability under realistic data volumes, reuse across use cases, and finally output interpretability.

5.2.1 Multiple Restrictions

The initial path accepted a single restriction per interaction. Real business requests, however, frequently combine several restrictions. Supporting only one restriction therefore limited how closely the solution matched real use.

Two designs were considered. The first would issue one LLM call per restriction and aggregate the results before execution, leaving the existing modules almost unchanged. Its main weakness is not runtime, since separate calls could in principle run in parallel. The weakness is that it presupposes the request has already been split into individual restrictions. Deciding where one restriction ends and the next begins is itself an act of interpretation, so this design would require a prior interpretation step to do the very separation it depends on, and it would also duplicate the prompt and domain description across calls. The second design issues a single LLM call that receives the complete input, identifies and decomposes all restrictions present, and returns an array

of candidate representations. This design was adopted: it performs separation and interpretation in one reasoning pass, avoids the duplicated context of repeated calls, and still leaves deterministic validation to operate on each restriction individually.

The extension changed the data contract along the workflow. The LLM now returns an array of candidate representations, even for a single restriction, and each element carries a short natural-language description of the restriction it represents. This array is internal to the pipeline. The user submits the request and receives an allocation, an error, or an explanation, without an intermediate step to confirm the interpretation. The description is an identification field only. It does not enter the formal structure expected by the engine and does not interfere with the four fields used for validation and assembly. Its sole purpose is to attribute feedback to the correct restriction in later stages, so that error messages identify which restriction failed without exposing technical indices. Making the array the only output form also lets every downstream stage handle input the same way: each check receives a list and iterates over it, with no special case for a single restriction.

The prompt's single-restriction rule was replaced by a dedicated decomposition section, and the remaining sections — validity criteria, error handling, output format, and examples — were adapted to operate on each restriction individually rather than on a single input. Decomposition involves two distinct questions: how many restrictions the request as a whole contains, and, within each restriction, how many limits it produces. The two are independent. Although separation logically comes first, the prompt addresses the within-sentence patterns first, since these are the less obvious case. Separation is treated immediately after.

The first question concerns patterns that arise within a single, well-formed sentence. A sentence the user intends as one restriction can legitimately produce more than one limit, and the clearest case is a range. "Each customer must receive between 2 and 5 products" is a single restriction in the user's terms, yet it expresses a minimum and a maximum at once. The LLM decomposes it into two limits sharing the same aggregation level and affected subset — a minimum of 2 and a maximum of 5, both per customer. A range is, in fact, one of the cases the initial path rejected, since it could not produce more than one limit from a single restriction. The LLM is now taught to recognize the pattern and decompose it. A conjunction, by contrast, does not create a second restriction: it enumerates accepted values within one condition, resolved as already described for value resolution. For a scalar attribute, "Customers from Madeira and the Azores" remains one restriction with two accepted values. For a list-valued attribute such as a brand column, "Coca-Cola and Colgate" is read as a single list containing both values. By contrast, "Coca-Cola or Colgate" yields separate lists — the distinction made available by the attribute type in the domain description. Distinguishing a range, which yields several limits, from a conjunction, which yields several values within one limit, is the core of this within-sentence handling.

The second question concerns how distinct restrictions are separated from one another. The convention presented to the user is to separate them with a full stop, which keeps each sentence a self-contained unit that the LLM processes independently. The UI was updated from the single-restriction guidance of the initial path to state this convention explicitly and to include an example with two restrictions in one input, as shown in Figure G.3 of Appendix G. Now that multiple

restrictions are accepted, inputs that were previously rejected must be interpreted correctly regardless of how the user separates them. Because users do not always follow the full-stop convention, and the solution must never produce a result that does not correspond to the complete intent, the LLM is additionally instructed to handle ambiguous separators as a fallback. When a semicolon, a comma, a slash, a plus sign, or a conjunction appears between what may be two restrictions, the LLM treats the marker as a separator only when objective indicators are present: different numeric limits, different aggregation levels, or clearly different subjects. For example, "At most 3 per customer and at least 50 per product" carries different limits and different dimensions on each side, so the conjunction separates two restrictions. By contrast, "Customers from Madeira and the Azores receive at most 2 products" shares one limit and one dimension throughout, so the same conjunction enumerates values within a single restriction. When these indicators are absent, the within-sentence patterns apply normally and the input is processed as a single restriction. The full decomposition section of the prompt is provided in Figure C.2 of Appendix C.

The attribute and value check then validates each candidate representation independently, iterating over the whole array. The internal checks were unchanged, each still operating on one restriction, and only the coordinating logic was adapted to iterate and accumulate results. Validated representations are then aggregated into a single set of limits in the form the engine already accepts: limits sharing an aggregation level are grouped together, and those on a different level form a separate entry. The aggregation also assigns a unique identifier to each restriction, so that distinct restrictions sharing the same operator and aggregation level remain separate rather than collapsing into one. The identifier of a limit was initially derived from its operator and aggregation level alone — a maximum per customer, for instance — so that two distinct restrictions sharing both produced the same identifier, and one silently overwrote the other when the limits were assembled. This error was invisible to per-restriction validation, since each candidate was individually well-formed, and surfaced only once the complete set was assembled against the engine. It was resolved by adding a per-restriction discriminator to the identifier, so that each restriction remains a separate limit regardless of any shared operator and level. The engine natively supports multiple rules per dimension, even though the initial path never produced more than one, so neither the engine nor the optimization required modification.

The workflow follows an all-or-nothing rule: if any restriction in the request fails at any stage of the orchestration, the whole request is stopped before reaching the engine. Executing only the restrictions that happened to pass would silently apply part of the user's intent and return a result that was not requested. A user who writes three restrictions expects the outcome to reflect all three. At each stage that can reject a restriction, from interpretation to deterministic validation, the request is not abandoned at the first failure: every failing restriction is reported. Each message is prefixed by the description of the restriction it concerns, so the user sees the complete set of problems at once and can correct and resubmit the whole request.

A structured set of 31 test cases was used to evaluate the behavior of the extension across representative scenarios, each run through the complete workflow as before. The cases are organized into the nine groups summarized in Table 5.5.

Table 5.5: Multi-restriction validation test groups.

Group	Cases	What it validates
A	2	Range decomposition on the same dimension
B	2	Restrictions across both dimensions
C	3	Three to five restrictions in one request
D	5	Mutually incompatible restrictions under varied separators
E	4	Conflicting restrictions on the same dimension
F	5	Spelling, casing, and conjunction or disjunction semantics
G	4	Requests mixing valid and unsupported restrictions
H	4	Values absent from the prepared data
I	2	Cross-entity attributes

The cases were designed to be harder than expected use. Most depart from the full-stop convention — conjunctions between restrictions of different dimensions, semicolons between rules with distinct limits, slashes and plus signs as dividers, continuous phrases without a full stop — on the rationale that if decomposition holds under these adverse conditions, it holds trivially when each restriction is a sentence ended by a full stop. The groups span the range of outcomes the extension must handle: requests that reach the engine, requests that are individually valid but jointly incompatible, and requests that must be stopped at interpretation or at one of the deterministic checks.

The 31 cases produced the expected outcome at the expected stage in the validation performed, with interpretation subject to the same run-to-run variability as the base workflow. The requests that reached the engine were resolved either with an allocation or, where the restrictions were jointly incompatible, with a correct infeasibility report; the rest were intercepted before execution. Behavior on patterns already validated for single restrictions — semantic mapping, orthographic robustness, list-valued and structured attributes — remained stable when those patterns appeared alongside others, since each restriction is processed independently after decomposition. The residual limitation is that interpretation of intent remains subject to the inherent ambiguity of natural language.

5.2.2 Special Cases

A business user may state a restriction in a form the initial path could not represent: that customers receive only products of a given type, that a stimulus reach everyone except a given region, or that the affected products be those that contain a given element. These were rejected at interpretation, under the same rule as any input that does not map onto the four fields of a limit.

That rule, however, hid a distinction the initial path never needed to make, because rejected inputs were not processed further. Some rejected restrictions are irrecoverable. “Customers with more than 3 children can receive at most 2 products”: the affected subset has nowhere to hold “more than 3”, since it carries an explicit value such as the number 3, not a comparison against it.

The same holds for a percentage or a count of elements. Each would need an operation the limit structure has no field for, and a user can always state another variation, creating an open space that no finite set of rules can close. Negation, exclusivity and containment are not like this. The LLM already interprets them as it interprets an ordinary filter — it identifies the attribute and resolves the value the same way — and the result is missing only a single deterministic step afterwards. These three are the patterns a fixed transformation can return to the valid limit form: the interpretation already produces the four fields, complete and well-formed, and only a deterministic operation over the data remains. This structural property is why they are the ones this extension makes available. The criterion for what counts as a valid limit does not change; what widens is the set of inputs that reach it.

What makes these three recoverable is clearest in the structured representation. Consider "Customers not from Madeira can receive at most 2 products." The LLM interprets this exactly as it would the ordinary restriction "Customers from Madeira can receive at most 2 products": aggregation level per customer, operator maximum, bound 2, affected subset the customer-region value Madeira. The four fields are complete and well-formed; the only addition is a marker, "negation:region", recording that one deterministic step remains. Nothing in the limit structure is bent — a transformation is simply deferred. This is the property the three patterns share and the irrecoverable cases lack: the interpretation already produces a valid limit, with a note that a fixed operation over the data must follow.

The prompt was given a section, between the decomposition rules and the error rules, that defines the three patterns. For each it states the trigger wording, the meaning, and the marker to add. Every other field is produced as for any restriction. The substance of the section is a set of distinctions. While these formulations were rejected, telling them apart was unnecessary. Now that they are accepted, the same surface wording can carry opposite meanings. A misclassification would produce a structurally valid limit that executes and returns the wrong result — the same concern that governs the rest of the solution. Three distinctions carry most of the weight, each taught with contrastive examples. The word "only" marks exclusivity when it restricts which products are assigned, but an ordinary maximum when it restricts how many: "Customers should receive only hygiene products" is exclusivity, whereas "Customers should receive only 3 products" is a maximum with bound 3. A negation inverts a filter in "Customers who are not from Madeira", but is an ordinary block with bound zero in "Customers from Madeira must not receive". And containment needs explicit words such as contain or include: "Products that contain Coca-Cola" is membership, whereas "Products from the Coca-Cola brand" is a normal filter, and "Products with the following values: Coca-Cola, Colgate, Nestlé" is also normal, being an explicit enumeration rather than containment. The full section, with its Portuguese trigger phrases, is given in Figure C.3 of Appendix C.

A special case is then validated in two stages, and their order is the safeguard. First it is checked like any other restriction: the attribute and values are matched against the prepared data, and an unmatched value is rejected here, before anything else. Only then does the transformation run. A special case therefore cannot bypass the validation that protects every restriction, and the

transformation never operates on values not confirmed against the data.

Each pattern then resolves against the data through a fixed deterministic step that draws its replacement values from those already present in the prepared data, so the resulting limit names explicit data values and requires no further matching. The exact transformation defined for each pattern, on scalar and on list-valued columns, is given in Appendix E.

One case is left unsupported by design: more than one distinct pattern within a single restriction. Supporting it would mean composing transformations — deciding which to apply first and feeding its result into the next. That ordering is itself an interpretation the LLM would have to make reliably, with the deterministic step then having to chain the transformations in the same order. Both the recognition and the resolution would grow considerably more complex, for a case far less common than a single special pattern. The decision was therefore to detect and refuse it rather than allow uncertain composition: a request such as "Customers who are not from Madeira should receive only products that contain Coca-Cola" combines negation, exclusivity and containment, and is returned as an error naming the three conflicting phrasings, so the user can understand what the request combined and reformulate it. The all-or-nothing rule from the previous extension holds here too: in a multi-restriction request, one failure stops the whole request before the engine.

A structured set of 52 cases evaluated the extension across representative scenarios. Each case was run through the complete workflow, and the validation confirms both the outcome and the stage at which it was resolved: supported cases reached the engine, while unsupported cases were stopped at the appropriate stage. The cases are organized into the groups in Table 5.6. A further 27 cases exercised the three patterns inside multi-restriction requests, confirming that the extension composes with the decomposition and aggregation mechanisms introduced earlier, including ranges, cross-dimension sets and infeasible combinations.

Table 5.6: Special-case validation test groups.

Group	**Cases**	**What it validates**
A	9	Recognition of each pattern, and cases previously rejected now resolved as special
B	8	Distinctions: similar wording with opposite classification
C	10	Conjunction and disjunction of values within a special case, on scalar and list columns
D	8	Patterns applied to list-valued columns
E	8	Combinations of two or more special patterns within one restriction, refused
F	9	Edge cases: absent values, non-list fall-through, multi-column targets, count-of-elements rejection

Across the 52 cases and the 27 multi-restriction cases, the supported patterns were transformed and executed as expected, while the unsupported and unmatched ones were intercepted at the appropriate stage. The cases that carry most weight are the look-alikes: restrictions whose wording

resembles a special case but is in fact ordinary, where a wrong classification would produce a valid limit with the wrong meaning. Because such an error yields a well-formed limit, it is the kind the deterministic checks cannot detect after the fact, which is exactly why these cases were the focus of the validation. The validation confirms that they were resolved by the stated criterion rather than by surface wording. The deterministic boundary that protects the initial path also applies here: a special case still stops at the limit-structure check if its aggregation level is unsupported, exactly as an ordinary restriction would. Pattern recognition is the more demanding judgment in this extension, and the contrastive distinctions are what make it reliable; the validation tested this directly and the criterion held.

5.2.3 Predefined Restrictions

A use case in practice carries business rules that apply to every optimization — capacity limits, distribution thresholds, minimum audiences for viability — conditions that must always hold regardless of what the user adds. These are the rules that keep a campaign operationally sound: maxima per customer to respect capacity, minima per customer to ensure each receives enough relevant content, and minima per product to guarantee sufficient distribution reach. Without predefined restrictions, the user would have to restate them in every interaction.

They are defined once in the use-case configuration, by the technical specialist who configures the use case, in the same module where data mappings and column definitions are established. They are part of the configuration of the use case, not something the business user writes at interaction. Each predefined restriction is stored in a dual format: the text in natural language, for presentation to the user, and a validated structured output ready to enter the pipeline directly without interpretation. Because the structured output is already complete and validated, there is no interpretation to perform. Invoking the LLM where no natural-language understanding is needed would add cost and risk without benefit. Predefined restrictions therefore bypass the interpretation step entirely and enter the pipeline in their prevalidated form.

The user sees the predefined restrictions at the start of the interaction and can keep, remove, or edit any of them. The available operations are shown in Figure G.2 of Appendix G. A restriction that is kept enters directly in its prevalidated form. One that is removed is excluded before execution. Editing is treated as removing the predefined restriction and writing a new one: only the new text enters the interpretation path, while the removed original is discarded. New restrictions written from scratch follow the full interpretation and validation path as in any other interaction. The cells through which restrictions are written, removed, and the optimization run are shown in Figure G.4 of Appendix G.

Predefined and user-written restrictions are merged into a single set, and the deterministic validation then checks both together for structural correctness, which for the predefined restrictions is a deliberate redundancy: their structured output is validated at the point it is written, but because that is done by hand in the configuration, running it through the same check guards against a configuration error at negligible cost. Execution then receives one list regardless of origin, and the response reports how many predefined and how many new restrictions were applied. This

extension depends on the ability to handle multiple restrictions, since it combines several sources into one set. The interpretation step, the deterministic validation, and execution all remain unchanged. Because this extension adds no new interpretation or validation logic, it was verified not by a new set of cases but across its discrete interaction paths. It routes prevalidated restrictions and reuses the multiple-restriction handling already tested. The verified paths were keeping, removing, editing, and combining predefined restrictions with new restrictions, each confirmed functional. It reinforces the principle that governs the solution throughout: the LLM is invoked only where interpretation is required, and deterministic handling is used everywhere else.

5.2.4 Full-Scale Data

The controlled path was validated on a reduced subset of the prepared data, appropriate for development and for campaigns whose business scope is itself defined by filters. A further question is what happens when that scope is expanded to the complete prepared data of a use case: millions of customers and hundreds of products. Together, these form an assignment space whose eligible customer–stimulus pairs, each a decision variable, can number in the billions. A linear optimization model holds all its variables, constraints and objective function in memory at once, and at this scale no single instance on the available hardware accommodates the full problem. Batch processing is therefore imposed rather than chosen. A fixed number of customers is processed as one optimization instance, and the complete customer universe is covered by running successive batches under identical conditions. The question is then whether partitioning the customers across batches affects the satisfaction of the restrictions, which depends on the kind of restriction.

A per-customer restriction is unaffected by construction. It constrains each customer independently of every other — a maximum or a minimum on the assignments a single customer receives — and every customer is processed whole, within one batch. Which other customers share that batch does not change what the restriction imposes on that customer, so partitioning cannot alter the outcome for any per-customer restriction. This is a structural guarantee, not an observation about the data.

A per-product restriction is different, and here satisfaction within each batch is conditional. A minimum per product — that a product reach at least a given number of customers — ranges over customers, who are divided across batches. Applied within each batch, it is satisfied only when every batch contains enough eligible customers for the product to meet the bound locally. Whether that condition holds is a property of the data, not of the architecture: it depends on each product's eligible customers being distributed densely enough that any batch of the sizes used contains a sufficient number of them. In the tested use case the condition held: eligibility across the customer population is broad enough that every product met its bound within each batch. The re-execution of the accumulated test cases over the production data confirmed this, with no per-product restriction failing for lack of local eligibility. This is therefore an empirical result under the conditions tested, not a property batch processing guarantees in general.

The remaining elements of an interaction are identical across batches: the affinity scores that drive the allocations are intrinsic to each customer-product pair and independent of which customers share a batch, and the domain description, the predefined restrictions and the user-written restrictions are the same for every batch. Within the conditions above, the batch size is then a computational parameter — changing it means changing one number in the configuration, with no effect on interpretation, validation or execution.

Two parts of the use-case configuration changed in moving from the controlled subset to the full data. The first is the set of filters applied to the data. The subset was deliberately reduced for development, where a smaller and stable volume kept the environment observable. At production scale that purpose no longer applies, and the only filter that remains is the number of customers per batch, which is a computational bound set by available memory, not a business choice. The second is the domain description. The information it exposes to the LLM must cover the values a user is likely to reference, and the full production data is far more diverse than the subset: a value that a restriction would naturally name could fall outside the examples the description exposed, leaving the products it identifies unreachable on that dimension. The number of examples exposed per attribute was set to cover the full set of values present at production scale, so that the values a user is most likely to reference are visible regardless of which part of the data is in view. This adjustment is to the configuration that feeds the domain description; the interpretation and validation logic was not touched. The predefined restrictions remain the universal rules of the operation, applied by default, now operating over the full data. Building the domain description over the production data is heavier than over the subset, since more data is scanned, but it remains a setup operation for the use case rather than part of every submitted restriction. The separation of setup from per-interaction execution established earlier therefore continues to hold: the larger data volume affects preparation, while repeated interactions do not rebuild the domain description.

The adaptations changed configuration, not processing logic, so confirming the production scale was a matter of re-running the test sets already defined for the earlier extensions rather than designing new coverage. The 185 cases accumulated across the controlled path and the preceding extensions — 75 from the end-to-end validation, 31 for multiple restrictions, 52 for special cases, and 27 for special cases within multi-restriction requests — were re-executed against the production data, with results consistent with those obtained on the controlled subset: every case either produced the expected output or was correctly resolved by the deterministic stages, and none produced a silently incorrect allocation. At this scale, the run-to-run variation of the interpretation step was likewise absorbed by the deterministic boundaries, with no observed case reaching execution in a form that violated the intended behavior. The re-execution confirms both that the consolidated domain description gives the LLM sufficient information across the full diversity of the production data, and that the per-product restrictions were satisfied within each batch.

Two boundaries remain. The batch size depends on the available hardware: a driver with more memory supports larger batches, with no other change. The soundness of batch processing for this use case rests on the two conditions just described, structural for per-customer restrictions and empirical for per-product ones. A restriction that ranges over the whole customer universe at once

cannot be decided within a single batch, since each batch sees only its share of customers. A global cap on the total number of assignments falls outside the limit structure itself, which expresses no restriction of that form, and is rejected like any other input the structure cannot represent. A maximum on the total number of customers a single product may reach is different: the structure expresses it, and the validation confirmed it on a single instance, where it holds correctly. Under batch processing, however, it is applied within each batch rather than across the whole universe, so the bound holds per batch rather than globally. The result is not incorrect — each batch enforces the restriction as specified — but it is not the global bound the restriction expresses. The system does not currently distinguish this case from an ordinary per-product restriction, and detecting or supporting it across batches is taken up as future work.

5.2.5 Use-Case Reuse

The personalization activities — leaflets, newsletters, and mailings — are distinct, and a solution meant for practical use cannot be bound to the one it was built around. With the controlled path established, extended for realistic input, and operating at full data scale, the remaining question was whether it could be reused in a different activity through configuration. A second use case was introduced to test this: the first concerned personalized leaflets, the second personalized newsletters. The two are deliberately far apart. The leaflet data carried both customer and stimulus attributes, including list-valued columns. The newsletter data has no customer attributes at all beyond the identifier, a much smaller catalog of stimuli, and no list-valued columns, so the containment pattern has nothing to apply to. The second use case is not a near-copy of the first but one that exercises the architecture where it has less to draw on, which tests it more sharply than a similar case would.

Integrating it was a matter of configuration. A use-case configuration was provided — the data, the column descriptions that ground the LLM in the new attributes, and the predefined restrictions expressing the operation's universal rules — and the use case was made selectable in the interface, as shown in Figure G.1 of Appendix G. Choosing it loads its configuration and predefined restrictions, and the orchestration draws the corresponding data when the pipeline runs. The deterministic logic — validation, assembly into limits, aggregation, and execution — was reused without modification, and the architecture did not change. Adapting to the new domain meant describing it in its configuration, not altering the shared pipeline.

Testing against this dataset produced cases the first use case never could, and they show concretely that the configuration governs behavior. A restriction filtering on a customer attribute — natural for a user, since the first use case allowed it — is correctly rejected here, because the configuration declares no customer attributes. The pipeline acts on what the configuration describes, not on what the first use case happened to contain, and so it neither invents a dimension that is absent nor fails ungracefully when one is. Interpretation is grounded the same way, and one case illustrates a convention the configuration carries beyond the column names. The column recording each stimulus's business priority runs on an inverted scale, where the highest priority carries the

lowest number, and a request for “high priority” is mapped to the correct value because the column’s explanatory note states this convention. Applying that note, mapping “high” to a number, asks more of the LLM than matching a term to a stored value. It is an interpretive step, and as such it falls within the same residual reliance on the LLM that holds throughout, rather than being removed by it. What the architecture contributes is where the convention lives: in the use-case configuration, as domain knowledge particular to this operation, which the LLM applies rather than inferring on its own. The value it produces is still matched against the column, which rejects any value the data does not contain. This is the separation of responsibilities as intended — the convention is declared in the configuration, and the LLM applies what is declared. The two cases are the same principle from two sides — the system acts on what the configuration declares, and interprets according to what the configuration describes.

A test set of 178 cases — built specifically for this use case, distinct from those re-run at full scale for the first — exercised the complete pipeline across every column of the new dataset and every type of restriction, individual and combined. The cases are organized into the groups summarized in Table 5.7.

Table 5.7: Second-use-case validation test groups.

Group	Cases	What it validates
A	6	Structure and dimension coverage
B	20	Attribute filtering across the stimulus columns
C	13	Semantic interpretation, including the inverted-scale convention
D	8	Rejection of unsupported requests
E	30	Special cases: negation and exclusivity
F	36	Linguistic robustness and dataset edge cases
G	65	Multi-restriction composition

Of the 178 cases, those expressible in the configured domain reached the engine, while requests filtering on customer attributes — natural for a user familiar with the first use case — were rejected here, because the configuration declares no customer attributes. The behavior at each stage matched the first use case, and no case produced a silently incorrect allocation. These rejections are themselves evidence that the pipeline acts on the configured domain rather than the original one. This is evidence of configuration-driven reuse for a configured use case, not of universal generalization. A use case whose data or rules fell outside what the configuration can express would require more than configuration. Within that boundary, however, it shows that the controlled layer is reached by describing a new domain, not by rebuilding the path.

5.2.6 Output Explanation

The controlled semantic layer addressed the problem this dissertation set out to solve: translating business restrictions stated in natural language into validated formal limits, without the technical

mediation this translation previously required. Within the wider process in which the solution operates, technical mediation remained necessary at a second point. The problem context described a cycle: a restriction is specified, the engine runs, and when execution fails or no feasible allocation exists, the outcome is reviewed, the specification is adjusted, and the cycle repeats. That review was also technical work, because the engine returns technical artifacts rather than a business explanation when no allocation is possible. This extension goes beyond the dissertation's central problem by addressing that second point, so that the same business user who states a restriction can also understand its outcome — natural language in, and natural language out — without per-interaction technical mediation at either stage. Figure 5.2 summarizes the decision flow across all outcomes, which the rest of this section describes.

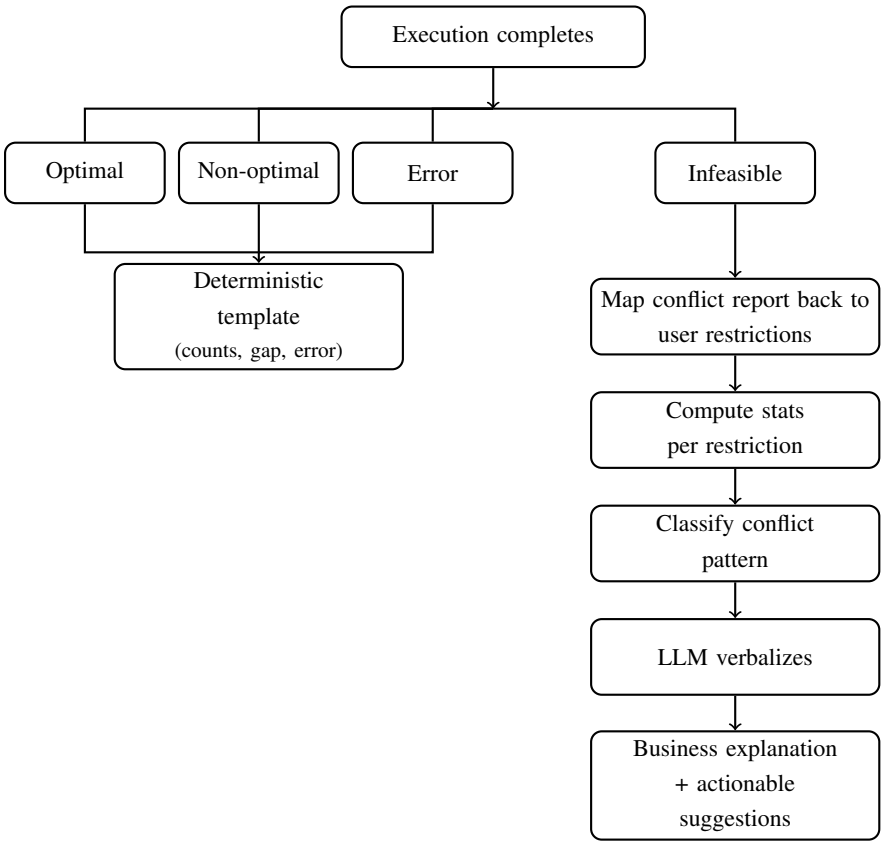


Figure 5.2: Decision flow for outcome explanation.

The engine returns one of a small number of outcomes, and the feedback was extended to explain each in the user's own language. When an allocation is produced, the information the user needs is well defined and computable directly from the result: how many assignments were made, over how many customers and stimuli, and how many restrictions were satisfied. The distinction between an optimal allocation and one found within the time budget without a proof of optimality matters in practice, since a campaign normally seeks the best allocation. This distinction is not visible in the result itself, so the feedback reports it explicitly. For a non-optimal result, it states that the allocation was found but not proven optimal, so the user knows the result is usable but not

guaranteed to be the best, and points to the fixed adjustment that can be made if a proven-optimal result is needed. A genuine execution error, distinct from this case, is reported as such. These outcomes are explained by deterministic templates, because in each the required information is fixed and known. The LLM is not involved.

The infeasible outcome is different, and it is where the extension does its real work. When no allocation can satisfy all restrictions at once, the solver computes what is termed an Irreducible Inconsistent Subsystem (IIS) — a subsystem of constraints that the solver identifies as mutually inconsistent — and writes it to a technical file. That file expresses the conflict as the optimization model sees it: lines naming individual assignment variables, often hundreds, with no reference to the business restrictions the user actually wrote. Figure H.1 of Appendix H reproduces a sanitized excerpt, and it makes the need for this extension concrete: nothing in it is interpretable without knowledge of the optimization model's internal form, which is precisely the knowledge the solution exists to make unnecessary.

The explanation of an infeasible outcome is built so that the reasoning about the conflict is done deterministically, and the LLM only puts it into words. The conflicting constraints reported by the solver are first mapped back to the user's stated restrictions, so that the conflict is expressed in terms of the restrictions the user recognizes rather than internal variables. For each restriction involved, the relevant quantities are then computed against the prepared data — how many entities fall within its scope, how many eligible pairs exist, and, for a minimum, how many entities fall below the required bound and by how much. From these quantities the nature of the conflict is classified before the LLM is ever called: whether a minimum exceeds the population available to satisfy it, whether two bounds are arithmetically incompatible, whether a blocking restriction prevents entities that another restriction requires to be served, or whether a single entity is the outlier that cannot meet a bound the rest can. These are the conflict modes the analysis distinguishes, organized around the lower- and upper-bound effects that make limits infeasible over the customer and stimulus dimensions; the classification follows from the computed quantities rather than from any judgment the LLM makes. When those quantities single out a restriction whose bound cannot be met even in isolation, as in a single-entity shortfall, that restriction is reported as the primary cause. When they do not, the analysis does not select one, and the response falls back to the general form described below. Only this structured result — the conflicting restrictions named in the user's terms, the quantities that prove the incompatibility, and the classified conflict mode — is passed to the LLM, which expresses it as a short explanation in business language together with concrete, actionable suggestions. The LLM receives no raw solver output and does not diagnose the cause. It verbalizes a conflict that has already been identified and quantified. The prompt that governs this verbalization is given in Figure C.4 of Appendix C.

This is the controlled principle of the input side applied in reverse. On the input side the LLM interprets and deterministic components validate; on the output side deterministic components analyze and the LLM verbalizes. In both directions the LLM operates inside boundaries set by deterministic logic. On the input side, deterministic validation decides whether a candidate representation may be executed. On the output side, deterministic analysis identifies what makes a set

of restrictions infeasible. The intelligence of the diagnosis is computational. The LLM contributes fluency, not judgment.

A concrete case shows what the explanation produces. A user states "Each product must reach at least 50 customers", to guarantee distribution across the catalog. If one product has only 14 eligible customers, no allocation can satisfy that requirement, and the solver returns infeasible. The deterministic analysis identifies that single product as the one below the bound, computes the gap between its eligible customers and the requirement, and classifies the conflict as a single-entity shortfall. The explanation then names the user's own restriction, states that one product cannot reach the required number because too few customers are eligible for it, and suggests either lowering the bound to what that product can meet or excluding it from the requirement — each suggestion tied to the restriction the user wrote and to a concrete value. What the user reads is a business explanation and a next step, not a set of conflicting variables. Figure H.2 of Appendix H shows the complete output for this case.

The same guarantee that governs the rest of the solution holds here: a usable message is returned for every outcome. The explanation of an infeasible result depends on the conflict being identified deterministically first. The analysis described above resolves the common case, in which the reported subset maps to the user's restrictions and the conflict can be stated over them. In the residual case, where the solver's conflict report does not isolate restrictions that map cleanly to what the user wrote, the LLM is not asked to guess, because a specific but wrong explanation is worse than a correct general one. In that case the feedback falls back to a deterministic message that lists the restrictions in play and directs the user to relax one of them. The response adapts accordingly, from a detailed explanation grounded in computed quantities to a general but correct statement of the situation. At no point is the user left without a response or given an explanation that the underlying analysis does not support.

The translation logic was added as a separate module, with a single new step in the orchestration between execution and the response to the interface. The interpretation prompt and the deterministic validation were left unchanged. Because every execution of the engine now passes through this step, the test sets of all the preceding extensions exercise it, each run producing the message appropriate to its outcome. A further set of 8 cases was added specifically to exercise distinct infeasibility patterns, including capacity conflicts across dimensions, a filter that reduces a population below a required minimum, infeasibility arising from the combination of restrictions rather than any one of them, and edge cases such as a minimum exceeding the available population. Across these, every outcome produced a message consistent with the actual result, and no case returned an explanation that misrepresented the conflict.

This extension completes the interaction loop and applies the controlled principle to the output side, but it is complementary to the central contribution rather than equal to it: the core of the work is interpreting and validating restrictions on the way in, and this extension makes the result of that work intelligible on the way out. The explanation speaks to the business user, who would otherwise depend on technical mediation to know why an allocation failed. It also helps the technical specialist, who reads a conflict stated in business terms faster than the raw solver file the

same outcome would otherwise require. It depends entirely on the structured information prepared deterministically before the LLM is invoked — without that preparation, there would be no sound basis on which to produce an explanation at all.

5.3 Discussion

The results in this chapter were obtained across successive development cycles, each validated as it was built. Seen as a whole, they support the following observations.

The validation exercised the solution against conditions representative of practical use, and the form those conditions take differs across the cycles. The initial controlled path was validated end-to-end over representative restriction categories, confirming that the path works end-to-end, from a natural-language restriction to a resolved outcome. Multiple restrictions and special cases were validated against diverse linguistic formulations — test cases spanning the ways a user phrases an intent. Predefined restrictions were validated against the interaction paths a user follows: keeping, removing, editing and combining. Full-scale data was validated against the volume of real production data, confirming that the workflow runs over it as over the subset. The second use case was validated against a different domain, confirming that the pipeline reaches it through configuration alone. Output explanation was validated against the range of outcomes the engine returns, confirming that each produces a faithful message. Each cycle answers a different question, so its validation takes a different form. What unites them is their nature: the evidence is technical and functional throughout, establishing that the solution does what it was designed to do across the conditions realistic use produces.

The contribution is the controlled solution itself, presented together with its progressive extensions as the result. The evidence establishes that it is sound, but it does not measure time saved, configuration effort, or campaign outcomes. Measuring that practical effect would require a different kind of evaluation, operational measurement or a user study, which falls outside this work and is taken up as future work.

One property runs through the whole solution and is stated here once: the interpretation of a restriction rests on the LLM, which can produce a representation that is well-formed yet does not match the user's intent. The design does not remove this risk entirely. Instead, it contains the classes of error that can be checked deterministically: structurally invalid, unsupported, or data-incompatible interpretations are prevented from reaching execution. Within the tested scenarios, no silently incorrect execution was observed. The reliability demonstrated here therefore rests not on perfect interpretation by the LLM, but on the deterministic components surrounding it and on the explicit representation of what was interpreted. The limitations specific to each extension were noted where they arise and are drawn together in the concluding chapter.

The chapter has demonstrated the technical and functional viability of the controlled semantic layer and its maturation toward realistic use: natural-language restrictions interpreted, validated and executed under control, extended to the conditions real use imposes — multiple restrictions,

controlled special cases, operational configuration, full-scale data and a second use case — and made intelligible on the way back.

Chapter 6

Conclusions and Future Work

This dissertation aimed to address the dependency on manual technical mediation in translating business restrictions, expressed in natural language, into the formal limits that an existing optimization engine requires for retail personalization. Through the design and validation of a controlled semantic layer, the work targeted this gap while preserving the rigor, validation and compatibility required by the optimization engine.

The initial motivation for this work arose from the way personalization decisions are currently supported. The optimization engine, Maestro, already resolves the customer–stimulus allocation, but it operates only on structured inputs; the scenario-specific element among these is the set of formal limits, which must be produced for each case. In the existing process, a business user would state a restriction in natural language and a technical specialist would translate it into the four elements of a limit, aligning each with the prepared data and the engine's input contract. This dependence introduced delay, reduced the autonomy of the users who best understood the intent, and exposed the process to a subtler risk: a limit can be formally valid while still encoding a meaning different from the one intended.

To address this, a controlled semantic layer was developed in which the responsibilities are kept separate. The LLM interprets a restriction and proposes a candidate structured representation; the domain description grounds that interpretation in the available attributes and domain values, while deterministic validation checks compatibility with the prepared data and with the engine's input contract; an assembly step constructs the corresponding limit; and Maestro, left unchanged, remains responsible for the formal optimization. The same controlled principle was later applied in reverse, so that when the engine returns an infeasible outcome, the conflict is identified, quantified and classified deterministically, and the LLM only verbalizes that analysis in business terms. Throughout, the LLM's interpretation was treated as a hypothesis to be verified, not as a final constraint.

The solution was developed incrementally and validated as it was built. A first development cycle established the end-to-end workflow as a working MVP, and successive extensions then adapted it to the conditions of realistic use: support for multiple restrictions within a single request, selected special cases resolved through fixed deterministic transformations, predefined restrictions

that bypass interpretation entirely, operation over production-scale data processed in batches, and a second, deliberately different use case integrated through use-case configuration without changes to the shared core pipeline. The evidence gathered across these cycles was technical and functional, based on validation sets using real domain content rather than synthetic inputs. It demonstrated that the supported restrictions were transformed into Maestro-compatible limits, that the deterministic checks prevented structurally invalid, unsupported or data-incompatible candidates from reaching execution, and that, within the tested scenarios, no silently incorrect execution was observed. This reliability was bounded by design, resting on the deterministic components surrounding the LLM rather than on the assumption of always-correct interpretation.

In these terms, the work answered the questions that guided it. It characterized the limitations of the current specification process as a dependency on manual technical mediation rather than as an impossibility; it showed how a controlled semantic layer can connect natural-language restriction specification with formal optimization while keeping responsibilities separated; and it demonstrated how supported business restrictions can be transformed into valid Maestro-compatible limits. On the evaluation of the solution, the two dimensions separate. Its robustness was established within the tested scenarios, and repeated use was shown to be practical once the construction of the domain description was separated from per-interaction processing. Its operational value, however — time saved, configuration effort, or the effective autonomy of business users — was not quantitatively measured, and is therefore left for future evaluation.

The contribution of this dissertation is not the use of a language model, but the controlled architecture in which it is placed. Through this separation of responsibilities, the work showed how business language can be brought closer to a formal optimization engine without delegating validation, decision or execution to the LLM. The LLM contributes flexible interpretation, the deterministic components contribute control, and the optimization engine retains responsibility for the allocation it already performs. What this arrangement establishes, within the tested scenarios, is concrete: a business restriction reaches the engine without a specialist formalizing each limit by hand, the specification stays explicit and open to inspection rather than hidden in technical choices, an infeasible outcome returns as an explanation the user can act on, and a new use case is reached by describing it in configuration rather than by rebuilding the path. These benefits are distinct from the operational value that continuous use might later show.

While the work established this capability, several limitations remain and define the boundaries of what the evidence supports. The most important limitation is that LLM interpretation may produce a well-formed and formally valid representation that does not match the intended meaning. Such plausible semantic mismatches cannot be eliminated by deterministic validation alone, since the resulting limit is well-formed; this residual risk is reduced by domain grounding and deterministic validation, and made more inspectable by the explicit representation of what was interpreted. The solution also covers only restrictions that can be expressed within the supported limit structure, and selected special cases were supported individually, while their composition within a single restriction was left outside the scope. The full-scale batching strategy was appropriate for the local restrictions of the tested use case, satisfied within each batch under the data

conditions observed. Restrictions requiring coordination across the whole customer universe at once would require a different execution strategy. Finally, the evaluation was technical and functional: it did not include a formal user study or a quantitative measurement of operational value.

These limitations point to the most relevant directions for future work. The composition of several special cases within a single restriction could be studied as a controlled extension, keeping the ordering of deterministic transformations explicit rather than delegating it to the LLM. A further direction is to strengthen the review of interpretations that are well-formed but semantically ambiguous, for example by presenting the interpreted representation to the user for confirmation before execution, extending the inspectability the structured representation already provides. Further work could evaluate configuration-driven reuse in structurally different use cases, where the domain description, supported attributes or restriction patterns differ more substantially from those tested here. A direction the present work points to as particularly relevant concerns global restrictions that range over the whole customer universe at once. These take two forms: a cap on the total number of assignments, which the limit structure does not express, and a maximum on the total number of customers a single product may reach, which the structure expresses but batch processing enforces only within each batch rather than globally. Studying how such restrictions could be specified and enforced — distinguishing those that can be resolved within a single batch from those that require coordination across batches — would extend the controlled layer to a class of restrictions that recurs in practice. Another extension would be to move from single-request specification to a controlled multi-turn interaction, allowing users to add, revise or remove restrictions through a conversational workflow while preserving deterministic validation before execution. Beyond these extensions, the natural continuation of the work is an operational evaluation, measuring time, effort, perceived autonomy, confidence and process integration beyond what a technical and functional demonstration can capture.

In terms of sustainable development, this dissertation relates most directly to Sustainable Development Goal (SDG) 9 (Industry, Innovation and Infrastructure), which the relevant guidance associates with engineering innovation in areas such as AI and data applied to industry, conducted responsibly. The work applies language-model interpretation to a business decision-support process through an architecture whose defining feature is deterministic control over that interpretation, making the application of such technology to business-restriction specification more transparent and verifiable. Its contribution to this goal is one of responsible innovation in an organizational process, direct in nature but limited in scope; possible indicators, such as the reduction of per-interaction technical mediation or the proportion of invalid requests stopped before execution, are presented as measures for future operational evaluation rather than as impact already demonstrated. A structured analysis is provided in Table I.1 of Appendix I.

In conclusion, the work showed that the interaction between business language and optimization can be mediated through a controlled semantic layer. It establishes a technically and functionally validated basis for making formal optimization more accessible, letting a business restriction be specified and its outcome reviewed through explicit, inspectable and controlled interaction.

References

AhmadiTeshnizi, A., Gao, W., and Udell, M. (2024). OptiMUS: Scalable optimization modeling with (MI)LP solvers and large language models. arXiv preprint arXiv:2402.10172.

Amershi, S., Weld, D., Vorvoreanu, M., Fourney, A., Nushi, B., Collisson, P., Suh, J., Iqbal, S., Bennett, P. N., Inkpen, K., Teevan, J., Kikin-Gil, R., and Horvitz, E. (2019). Guidelines for human-AI interaction. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–13.

Androutsopoulos, I., Ritchie, G. D., and Thanisch, P. (1995). Natural language interfaces to databases: An introduction. *Natural Language Engineering*, 1(1):29–81.

Bommasani, R., Hudson, D. A., Adeli, E., Altman, R., Arora, S., von Arx, S., Bernstein, M. S., Bohg, J., Bosselut, A., Brunskill, E., Brynjolfsson, E., Buch, S., Card, D., Castellon, R., Chatterji, N., Chen, A., Creel, K., Davis, J. Q., Demszky, D., Donahue, C., Doumbouya, M., Durmus, E., Ermon, S., Etchemendy, J., Ethayarajh, K., Fei-Fei, L., Finn, C., Gale, T., Gillespie, L., Goel, K., Goodman, N., Grossman, S., Guha, N., Hashimoto, T., Henderson, P., Hewitt, J., Ho, D. E., Hong, J., Hsu, K., Huang, J., Icard, T., Jain, S., Jurafsky, D., Kalluri, P., Karamcheti, S., Keeling, G., Khani, F., Khattab, O., Koh, P. W., Krass, M., Krishna, R., Kuditipudi, R., Kumar, A., Ladhak, F., Lee, M., Lee, T., Leskovec, J., Levent, I., Li, X. L., Li, X., Ma, T., Malik, A., Manning, C. D., Mirchandani, S., Mitchell, E., Munyikwa, Z., Nair, S., Narayan, A., Narayanan, D., Newman, B., Nie, A., Niebles, J. C., Nilforoshan, H., Nyarko, J., Ogut, G., Orr, L., Papadimitriou, I., Park, J. S., Piech, C., Portelance, E., Potts, C., Raghunathan, A., Reich, R., Ren, H., Rong, F., Roohani, Y., Ruiz, C., Ryan, J., Ré, C., Sadigh, D., Sagawa, S., Santhanam, K., Shih, A., Srinivasan, K., Tamkin, A., Taori, R., Thomas, A. W., Tramèr, F., Wang, R. E., Wang, W., Wu, B., Wu, J., Wu, Y., Xie, S. M., Yasunaga, M., You, J., Zaharia, M., Zhang, M., Zhang, T., Zhang, X., Zhang, Y., Zheng, L., Zhou, K., and Liang, P. (2021). On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*.

Calvanese, D., Dumas, M., Laurson, Ü., Maggi, F. M., Montali, M., and Teinemaa, I. (2018). Semantics, analysis and simplification of DMN decision tables. *Information Systems*, 78:112–125.

Fuchs, N. E., Kaljurand, K., and Kuhn, T. (2008). Attempto Controlled English for knowledge representation. In *Reasoning Web*, volume 5224 of *Lecture Notes in Computer Science*, pages 104–124. Springer.

Hillier, F. S. and Lieberman, G. J. (2010). *Introduction to Operations Research*. McGraw-Hill, 9 edition.

Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y. J., Madotto, A., and Fung, P. (2023). Survey of hallucination in natural language generation. *ACM Computing Surveys*, 55(12):1–38.

Katsogiannis-Meimarakis, G. and Koutrika, G. (2023). A survey on deep learning approaches for text-to-sql. *The VLDB Journal*, 32:905–936.

Lepenioti, K., Bousdekis, A., Apostolou, D., and Mentzas, G. (2020). Prescriptive analytics: Literature review and research challenges. *International Journal of Information Management*, 50:57–70.

Liang, P., Jordan, M. I., and Klein, D. (2013). Learning dependency-based compositional semantics. *Computational Linguistics*, 39(2):389–446.

Lin, S., Hilton, J., and Evans, O. (2022). TruthfulQA: Measuring how models mimic human falsehoods. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics*, pages 3214–3252.

National Institute of Standards and Technology (2023). Artificial intelligence risk management framework (AI RMF 1.0). Technical Report NIST AI 100-1, National Institute of Standards and Technology. doi:10.6028/NIST.AI.100-1.

Nuseibeh, B. and Easterbrook, S. (2000). Requirements engineering: A roadmap. In *Proceedings of the Conference on the Future of Software Engineering*, pages 35–46. ACM.

Popescu, A.-M., Armanasu, A., Etzioni, O., Ko, D., and Yates, A. (2004). Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING)*, pages 141–147, Geneva, Switzerland.

Power, D. J. and Sharda, R. (2007). Model-driven decision support systems: Concepts and research directions. *Decision Support Systems*, 43(3):1044–1061.

Quamar, A., Özcan, F., Miller, D., Moore, R. J., Niehus, R., and Kreulen, J. T. (2022). Natural language interfaces to data. *Foundations and Trends in Databases*, 11(4):319–414.

Ramamonjison, R., Yu, T. T., Li, R., Li, H., Carenini, G., Ghaddar, B., He, S., Mostajabdaveh, M., Banitalebi-Dehkordi, A., Zhou, Z., and Zhang, Y. (2023). NL4Opt competition: Formulating optimization problems based on their natural language descriptions. In *Proceedings of Machine Learning Research*, volume 220 of *NeurIPS 2022 Competition Track*, pages 189–203.

Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Zettlemoyer, L., Cancedda, N., and Scialom, T. (2023). Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Scholak, T., Schucher, N., and Bahdanau, D. (2021). PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9895–9901.

Sprague, R. H. (1980). A framework for research on decision support systems. In Fick, G. and Sprague, R. H., editors, *Decision Support Systems: Issues and Challenges*, pages 5–22. Pergamon Press, Oxford.

Wang, B., Shin, R., Liu, X., Polozov, O., and Richardson, M. (2020). RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578.

Xiao, Z., Xie, J., Xu, L., Guan, S., Zhu, J., Han, X., Fu, X., Yu, W., Wu, H., Shi, W., Kang, Q., Duan, J., Zhong, T., Yuan, M., Zeng, J., Wang, Y., Chen, G., and Zhang, D. (2025). A survey of optimization modeling meets LLMs: Progress and future directions. arXiv preprint arXiv:2508.10047.

Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J., Li, I., Yao, Q., Roman, S., Zhang, Z., and Radev, D. (2018). 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*, pages 3911–3921.

Zettlemoyer, L. S. and Collins, M. (2007). Online learning of relaxed CCG grammars for parsing to logical form. In *Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 678–687.

Appendix A

Restriction Formalization Examples

Table A.1: Example restrictions and structured representations.

Example	Natural-language restriction	Structured representation
A.1 Simple per-customer maximum	Each customer can receive at most 3 products.	Level: per customer. Customer condition: all eligible customers. Stimulus condition: all eligible stimuli. Operator: maximum ($\leq$). Bound: 3.
A.2 Per-stimulus minimum	Each product must be assigned to at least 100 customers.	Level: per stimulus. Customer condition: all eligible customers. Stimulus condition: all eligible stimuli. Operator: minimum ($\geq$). Bound: 100.
A.3 Customer subset restriction	Customers in the VIP segment must receive at least 1 product.	Level: per customer. Customer condition: segment in {VIP}. Stimulus condition: all eligible stimuli. Operator: minimum ($\geq$). Bound: 1.
A.4 Stimulus subset restriction	Each customer can receive at most 1 hygiene product.	Level: per customer. Customer condition: all eligible customers. Stimulus condition: product category in {hygiene}. Operator: maximum ($\leq$). Bound: 1.
A.5 Combined customer and stimulus subset restriction	Customers from Madeira or the Azores can receive at most 1 hygiene product.	Level: per customer. Customer condition: region in {Madeira, Azores}. Stimulus condition: product category in {hygiene}. Operator: maximum ($\leq$). Bound: 1.

Note: In the terminology used in the main text, the affected subset is shown here as its customer and stimulus conditions.

Appendix B

Domain-Description Construction

The domain description applies additional rules to specific attribute formats, determining how each attribute's type is classified.

String attributes containing codes, identifiers or textual categories are classified through a text-versus-number rule: if more than half of the characters of a representative value from the column are letters, the attribute is treated as text. A single value is sufficient for this decision because the values within a column share the same type and format, so the classification does not depend on inspecting them all. Text-like attributes are then handled as categorical or high-cardinality text attributes according to their number of unique values. For example, an email attribute is classified as text, but because it contains a large number of unique values, only representative examples are included in the domain description rather than the full set of values.

List-valued attributes are identified explicitly, since restrictions may refer to whether a value is contained within a list rather than equal to a single value. When list elements are short, complete example lists are included. When lists are longer, only the first K elements of each example list are shown. The objective is to expose the structure and contents of the list without introducing large volumes of repetitive information, since the LLM only requires enough examples to understand the attribute format and expected values.

The effect of this type-based selection on input size, compared with an unselective description, is reported in Table B.1.

Table B.1: Effect of type-based domain-description construction on input size.

Domain-description strategy	**Input tokens**	**Reduction**
Unselective description	31,223	—
Type-based description	7,994	74.4%

Note: The unselective description assumes a fixed 100 examples for every attribute, regardless of type or cardinality. The type-based description corresponds to the implemented setting: low-cardinality categorical attributes expose all their distinct values (up to $M = 100$), other attributes expose $N = 15$ examples each, and list-valued examples expose up to $K = 5$ elements per list.

Appendix C

Prompt Structure

VALID CONSTRAINTS
A valid constraint must:
- Be a simple aggregation (sum) of assignment variables
- Compare that sum to an explicit absolute numeric limit (non-negative integer)
- Be directly computable without transformations, comparisons, or distinct counts
- Be defined per customer or per stimulus
- Contain exactly one constraint

INVALID INPUTS & ERRORS
Return error when the constraint cannot be expressed directly in the required output format, either because:

1. Required information is absent:
   - no per-entity dimension (global constraints)
   - no explicit numeric limit (“algumas”, “muitas”)
   - objective functions (“maximizar”, “minimizar”)
2. Producing the output would require a transformation not available in the format:
   - counting elements inside a field value (“mais do que X”)
   - threshold or comparison filters (“acima de”, “superior a”)
   - percentage-based or relative limits (“50%”, “dobro”, “metade”)
   - any condition that depends on the dataset content to determine the limit or filter

   Also reject when no column in the domain description represents the intent.

Do not approximate: reject any request that cannot be expressed exactly with explicit values in the required format.
Error format: {“error”: “Erro: <razão breve>”}

Figure C.1: Prompt structure for valid and invalid constraints.

CONSTRAINT DECOMPOSITION
The user request may contain one or more constraints.

Processing order:
1. First, identify and separate all distinct constraints in the request
   - the user is expected to separate constraints using "."
2. Then, process each constraint independently, as if it were the only one in the request
   - each sentence is an independent unit
   - never propagate or share affected filters across sentences

Within-sentence patterns:
1. Single constraint
   - one rule with dimension, limit, and optional filter
2. Range expressions (e.g. "entre X e Y", "de X a Y", "mínimo X máximo Y")
   - decompose into 2 constraints: min X + max Y
   - both share the same dimension and affected subset
3. Conjunctions ("e", "ou")
   - enumerate values within the same filter of one constraint
   - non-list columns: "A e B" and "A ou B" both produce multiple values in the same key (OR semantics)
   - list columns: "A e B" → single list [A, B] (together);
     "A ou B" → separate lists [A], [B] (alternatives)

Fallback — ambiguous separators:
Symbols (";", ",") or conjunctions ("e") may appear in contexts where they separate distinct constraints rather than connecting values. When this occurs, evaluate the context:
- Only treat as constraint separator when clearly distinct rules:
  • different numeric limits on each side
  • different dimensions (customer vs stimulus)
  • clearly different constraint subjects
- If the context does not clearly indicate distinct rules, apply within-sentence patterns as normal

Representative distinctions:
- "Ofertas de A e de B no máximo N por cliente" → single constraint
- "Máximo N por cliente e mínimo M por oferta" → two constraints
- "Máximo N por entidade. Mínimo M por item." → two constraints
- "Ofertas de X entre N e M" → two constraints (range: min N + max M, same affected)

Output contract:
- Always return a JSON array, even for a single constraint
- Each element includes a "description" field: a short sentence in Portuguese identifying the individual restriction
- If some constraints are valid and others invalid, include both valid structured objects and error objects in the same array

Figure C.2: Prompt structure for constraint decomposition.

SPECIAL CASES
Some constraints fit the required output format but require additional processing downstream (deterministic validation step). When you identify one of these patterns, include a "special" field (format: "type:column_name") alongside all standard fields.

The column_name after ":" must be one of the keys in affected — it indicates which column(s) the special operation applies to. For multiple columns of the same type, use comma separation: "membership:column1,column2".
All other fields remain the same.

- Exclusive → "special": "exclusive:column_name"
  • The user wants only the specified values — the deterministic validation step blocks everything else.
  • Typically without an explicit numeric limit (limit 999999, a sentinel set above the number of eligible assignments any single aggregation unit can receive, so that it never binds and acts as no maximum; limit_type "max").

- Negation → "special": "negation:column_name"
  • The user defines what to exclude — the deterministic validation step inverts to the complement set.

- Membership → "special": "membership:column_name"
  • The user wants values containing a specific element — the deterministic validation step expands to matching entries.

Note: Identifying a special case only determines the "special" field.
All other prompt rules remain fully in effect.

Important distinctions:
Each group below shows patterns that look similar but produce fundamentally different outputs. Use the stated criterion to distinguish — small word differences change the result entirely.

1. Exclusive vs normal constraint
   Criterion: exclusive applies only when there is no explicit numeric limit — the restriction is on what type of assignments, not how many. When a numeric limit is present with "só"/"apenas", it means maximum (not exclusive, not equals).
   - "Clientes só devem receber N ofertas" → normal (limit_type = max, limit = N)
   - "Clientes só devem receber ofertas de X" → exclusive (limit 999999)
   - "Ofertas só de X devem ser no máximo N por cliente" → normal with filter

2. Negation vs explicit block
   Criterion: negation applies when the constraint is defined by what to exclude — values must be inverted. A direct filter with "não devem receber" is an explicit block (limit = 0), not negation.
   - "Clientes que não são de X" → negation
   - "Clientes de X não devem receber" → normal (limit = 0)

3. Membership vs direct filter
   Criterion: membership requires explicit containment words (e.g. "contenham", "incluam", "englobem"). All other references ("de X", "da X", "com os seguintes") are normal filters, regardless of column type.
   - "Ofertas que contenham X" → membership
   - "Ofertas de X" / "da X" → normal with filter
   - "Ofertas com os seguintes valores: X, Y, Z" → normal (explicit enumeration)
   The deterministic validation step handles list matching automatically for normal filters — membership is only needed for containment semantics.

Figure C.3: Prompt structure for special cases.

OUTPUT TRANSLATION
You explain why a customer-offer allocation failed, to business users.

CONTEXT
A system assigns offers to customers, maximizing affinity scores subject to user-defined constraints. The optimizer could not satisfy all constraints simultaneously. You receive pre-computed statistics showing what each constraint requires vs. what is available.

RULES
1. Write in Portuguese (Portugal). Address the user directly.
2. No technical jargon: never say "IIS", "LP", "bounds", "solver", "infeasible", or internal constraint names.
3. Be concise: max approximately 25 words per sentence.
4. Quote the user's exact constraint descriptions.
5. Use numbers from the data to make the conflict concrete.
6. Each suggestion must quote WHICH constraint to change and include a concrete number or action.
7. Only suggest changes that fully resolve the problem. Verify each suggestion against ALL constraints.
8. Only suggest actions the user can take: Reduzir, Aumentar, Remover, or Reescrever a constraint, or Excluir entities from the input data.

STRUCTURE
Part 1 — Explanation (3–5 sentences):
Quote conflicting constraints, explain WHY incompatible, show with one key number.

Part 2 — Suggestions:
"Considere as seguintes sugestões:" then 1–2 bullets with action verb + quoted constraint + concrete change.

EXAMPLES (abbreviated)
Example 1 — Single entity bottleneck:
"A restrição 'Cada oferta vai para pelo menos 80 clientes' não pode ser cumprida. A oferta 12345 é a única com menos de 80 clientes elegíveis (tem apenas 5).
Considere as seguintes sugestões:
• Reduzir 'Cada oferta vai para pelo menos 80 clientes' para 5 ou menos.
• Reescrever (. . .) excluindo a oferta 12345."

Example 2 — Arithmetic conflict:
"As restrições 'Cada cliente recebe pelo menos 5 ofertas' e 'Cada oferta vai para no máximo 20 clientes' são incompatíveis. Com 1.000 clientes e 200 ofertas, a primeira exige 5.000 alocações mas a segunda permite no máximo 4.000.
Considere as seguintes sugestões:
• Reduzir 'Cada cliente recebe pelo menos 5 ofertas' para 4.
• Aumentar 'Cada oferta vai para no máximo 20 clientes' para pelo menos 25."

Example 3 — Structural contradiction:
[Blocking rule contradicts a minimum on the opposite dimension.]

Example 4 — Systemic:
[Average eligible per entity is below the minimum required.]

Figure C.4: Prompt structure for output translation.

Appendix D

Model Comparison Analysis

The choice of model for the interpretation step was decided by running the candidate models on the same 75 validation cases, holding the prompt, the domain description and the validation set constant.

Two candidates failed in ways that made them unsuitable for the task. GPT-5-nano produced unparseable output in nearly a third of calls — structurally invalid rather than merely incorrect — and ran several times slower than the others in the evaluation runs. GPT-4o-mini lost components when a restriction combined multiple filters, and consistently selected a similarly-named but incorrect attribute when two columns shared the same domain.

Among the remaining candidates, higher cost did not translate into higher capability. GPT-5.1, at five times the input cost of GPT-5-mini, scored lower: it returned multiple values where one was expected and silently skipped filters rather than reporting an error.

GPT-5-mini, GPT-4.1-mini and GPT-5.1 scored consistently high across categories A–H, with isolated failures in the earlier categories. They diverged mainly on the advanced cases of category I, where the gap between them was widest, and it was performance there, rather than on the common restrictions, that distinguished them. The full breakdown of results by category and model is given in Table D.1.

Table D.1: Results by category and model.

Category	**Cases**	**5-nano**	**4o-mini**	**5-mini**	**4.1-mini**	**5.1**
A	6	5	6	6	6	6
B	8	5	6	8	8	8
C	5	3	4	5	5	5
D	9	6	9	9	9	8
E	9	4	6	8	7	7
F	4	2	3	4	4	4
G	3	2	2	3	3	3
H	8	5	7	8	8	7
I	23	11	12	22	15	18
Total	**75**	**43**	**55**	**73**	**65**	**66**

Appendix E

Special-Case Transformations

Each of the three recoverable special-case patterns — negation, exclusivity and containment — defers a single deterministic step until after interpretation. This step resolves the pattern against the prepared data, and what it does depends on whether the attribute is scalar or list-valued. In every case it draws its replacement values from those present in the prepared data, so its output references only values the data already contains.

For the negation "Customers not from Madeira can receive at most 2 products", the deterministic step replaces "Madeira" with the other customer regions present in the data, so the affected subset becomes "Azores" or "Algarve" or any region that is not "Madeira". For exclusivity, "Customers should receive only hygiene products" becomes a pair of limits from one interpreted restriction. The first keeps the hygiene category in the affected subset, with its bound raised above the largest number of assignments any single aggregation unit could receive, so that it never binds and acts as no maximum at all. The second carries every other category present in the data with bound zero, so that only hygiene products may be assigned. For containment on a list-valued column, "Products that contain Coca-Cola" replaces the single element "Coca-Cola" in the affected subset with every stored brand list that includes it — "[Coca-Cola, Colgate]", "[Coca-Cola, Nestlé]", and so on — so the limit names explicit data values rather than an element to search for.

Whether an attribute is scalar or list-valued changes what each transformation does. Negation and exclusivity operate on the exact stored value: on a scalar column, negating "Madeira" excludes that value and admits the rest; on a list-valued column, negating the stored list "[Coca-Cola]" excludes that exact combination, so a list such as "[Coca-Cola, Colgate]" is a different value and remains. Containment is defined only for list-valued attributes, where it expands a single element into the stored lists that include it. The marker determines which operation applies. Without it, the same value on the same column would be ambiguous between these results. A marker that names a column which is not list-valued, where containment has no meaning, falls through to an ordinary filter rather than failing. When more than one column carries the same transformation, the marker lists them, and each is transformed independently.

Appendix F

Validation Test Catalog

Table F.1: Distribution of validation cases across pipeline stages for the selected model.

Stage	Resolution	Cases
Interpretation	Correctly rejected as unsupported	17
Attribute and value check	Stopped — referenced value absent from prepared data	7
Limit-structure check	Stopped — attribute cannot enter a limit	1
Execution	Executed by the engine	50
Total		**75**

Note: The distribution reflects the stage at which each of the 75 cases is resolved. Each case was designed to exercise a specific behavior: some to be rejected at interpretation, some to be stopped by a deterministic check, and the rest to pass through to execution. Among the 50 cases that reach execution, most are supported restrictions that resolve into an allocation. A smaller number are deliberately conflicting requests used to verify that the engine detects joint infeasibility, of which 3 were reported infeasible and none returned an execution error. Run-to-run variability in the interpretation step does not change which cases reach execution, since invalid, unsupported or data-incompatible candidates are intercepted before that point.

Appendix G

User Interface

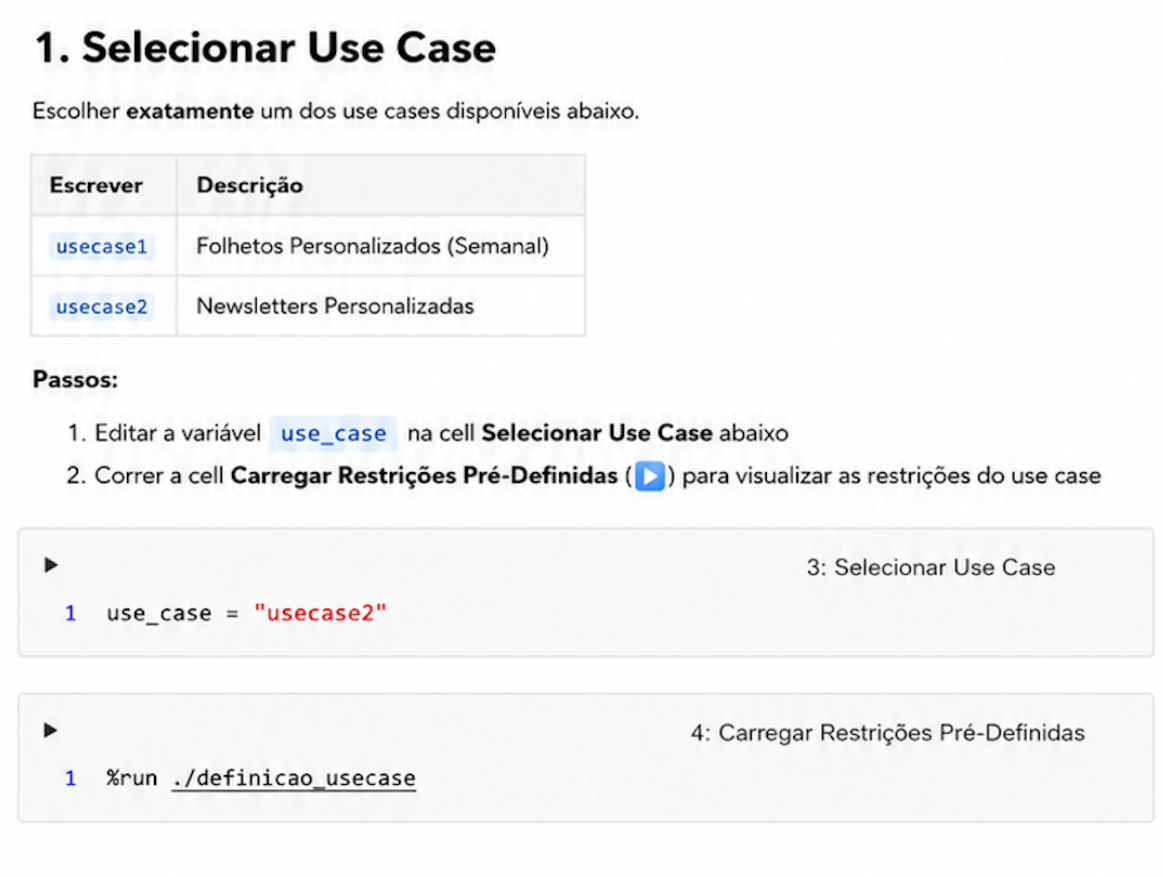


Figure G.1: Use-case selection.

2. Definir Restrições

Restrições Pré-Definidas

As restrições pré-definidas do use case aparecem numeradas na cell acima. Pode:

- **Manter todas** — deixar `apagar_restricoes = []`
- **Remover** — indicar os números em `apagar_restricoes` - ex: `[1,2,3]`
- **Editar** — remover a pré-definida (adicionando o número a `apagar_restricoes`) e escrever a versão alterada em `novas_restricoes = """ """`

Figure G.2: Interaction with predefined restrictions.

Adicionar Novas Restrições

Escrever **uma ou mais** regras na variável `novas_restricoes` , em **português**.

O sistema identifica automaticamente as colunas e valores do dataset — basta descrever o que pretende. Mas quanto mais específica a descrição (colunas/valores), mais preciso o resultado.

Exemplos:

- `Cada cliente pode receber no máximo 5 ofertas`
- `Clientes da madeira devem receber exatamente 3 ofertas`
- `Máximo 5 ofertas por cliente. Cada oferta vai para pelo menos 50 clientes.`

Orientações:

- Cada restrição deve indicar: **quantas ofertas por cliente** ou **quantos clientes por oferta**
- Pode especificar características dos clientes ou das ofertas para filtrar
- Deve incluir um limite numérico ("máximo 5", "exatamente 3", "pelo menos 2")
- Para múltiplas restrições, separar com ponto final ou mudança de linha

Depois correr a cell **Correr Pipeline** (▶️).

Figure G.3: Guidance for writing new restrictions.

Figure G.4: UI section for restriction specification and optimization execution.

Appendix H

Solver Output and Translation Example

```
Model optimization_model
\ LP format - for model browsing. Use MPS format to capture full model detail.
Maximize

Subject To
 min_stimulus_id_S007_0_c0: allocate_('C10201',_S007)
   + allocate_('C10455',_S007)
   + allocate_('C10712',_S007)
   + allocate_('C11038',_S007)
   + allocate_('C11247',_S007)
   + allocate_('C11589',_S007)
   + allocate_('C11602',_S007)
   + allocate_('C12044',_S007)
   + allocate_('C12198',_S007)
   + allocate_('C12367',_S007)
   + allocate_('C12501',_S007)
   + allocate_('C12884',_S007)
   + allocate_('C13005',_S007)
   + allocate_('C13291',_S007) >= 50
 min_customer_id_C10201_1_c1: allocate_('C10201',_S001)
   + allocate_('C10201',_S002)
   + allocate_('C10201',_S003)
   + allocate_('C10201',_S004)
   + allocate_('C10201',_S005)
   + allocate_('C10201',_S007) >= 1
Bounds
 0 <= allocate_('C10201',_S007) <= 1
 0 <= allocate_('C10455',_S007) <= 1
 0 <= allocate_('C10712',_S007) <= 1
 0 <= allocate_('C11038',_S007) <= 1
 0 <= allocate_('C11247',_S007) <= 1
 0 <= allocate_('C11589',_S007) <= 1
 0 <= allocate_('C11602',_S007) <= 1
 0 <= allocate_('C12044',_S007) <= 1
 0 <= allocate_('C12198',_S007) <= 1
 0 <= allocate_('C12367',_S007) <= 1
 0 <= allocate_('C12501',_S007) <= 1
 0 <= allocate_('C12884',_S007) <= 1
 0 <= allocate_('C13005',_S007) <= 1
 0 <= allocate_('C13291',_S007) <= 1
End
```

Figure H.1: Sanitized excerpt from the solver IIS file for an infeasible outcome.

A restrição “Cada oferta vai para pelo menos 50 clientes” não pode ser cumprida. A **oferta S007** é a única com menos de **50** clientes elegíveis (tem apenas **14**).

Considere as seguintes sugestões:

• **Reduzir** “Cada oferta vai para pelo menos 50 clientes” para **14** ou menos.

• **Reescrever** “Cada oferta vai para pelo menos 50 clientes” excluindo a oferta **S007** (a única abaixo do limite).

Note: The system operates in Portuguese and uses “oferta” as the generic business term for the stimulus dimension; quoted text reflects the user’s own formulation.

Figure H.2: Translated explanation presented to the user for the infeasible outcome of Figure H.1.

Appendix I

Sustainable Development Goals Analysis

Table I.1: Sustainable Development Goals analysis.

SDG	Goal / Target	Contribution	Possible performance indicators and metrics
SDG 9 — Industry, Innovation and Infrastructure	**Target 9.5** — Enhance scientific research and upgrade the technological capabilities of industrial sectors	• Development of a controlled semantic layer between business language and formal optimization. • Application of language-model interpretation under deterministic validation and formal execution control. • Support for more transparent, inspectable and verifiable AI-assisted interaction with optimization systems. • Preservation of the existing optimization engine as the component responsible for formal allocation.	• Proportion of restriction specifications processed without per-interaction technical mediation. • Proportion of invalid, unsupported or data-incompatible requests stopped before execution. • Number of configured use cases supported by the shared core pipeline. • Average per-interaction processing time after domain-description setup. • User-perceived clarity, confidence and autonomy in specifying restrictions.