Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Retrieval-Assisted Instantiation of Natural-Language Optimization Problems

Soroush Vahidi

sv96@njit.edu

New Jersey Institute of Technology

Research Article

Keywords: natural language processing, optimization modeling, knowledge representation, information retrieval, semantic grounding, intelligent information systems

Posted Date: July 27th, 2026

DOI: https://doi.org/10.21203/rs.3.rs-10463941/v1

License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: No competing interests reported.

Retrieval-Assisted Instantiation of Natural-Language Optimization Problems

Soroush Vahidi[1*]

[1*]Ying Wu College of Computing, New Jersey Institute of Technology, University Heights, Newark, 07102-1982, NJ, USA.

Corresponding author(s). E-mail(s): sv96@njit.edu (ORCID: 0000-0003-1934-6282);

Abstract

Many engineering optimization problems are first described in natural language and only later translated into formal mathematical models, but this translation remains difficult to automate reliably. Rather than targeting full solver-ready generation, this paper studies a narrower intermediate task: retrieval-assisted optimization problem instantiation, viewed as a knowledge-processing problem of grounding natural-language numeric evidence into a structured, typed optimization schema. The contribution is a transparent and reproducible framework that first retrieves the most compatible optimization schema from a fixed catalog and then deterministically instantiates that schema's scalar parameters from numeric evidence in the text, combining lexical schema retrieval, rule-based numeric extraction, coarse type inference, and deterministic compatibility-based slot assignment. We evaluate the framework on the NLP4LP benchmark of natural-language optimization problems, together with complementary structural and solver-backed validation subsets. Schema retrieval is already strong across query variants, with term frequency–inverse document frequency (TF–IDF) retrieval achieving the best average top-1 schema accuracy (Schema R@1 = **0.9094** on original-text queries). The strongest non-oracle method reaches an InstantiationReady score of **0.5287**, while an oracle control with perfect schema retrieval reaches only **0.5680**: this modest oracle gain shows that the main remaining bottleneck is downstream number-to-slot grounding rather than schema retrieval itself. A numeric-type compatibility ablation supports this conclusion: correcting how float-valued slots are matched to extracted numeric mentions yields large improvements in type consistency and query-level readiness. Restricted structural and solver-backed subset studies further suggest downstream optimization-oriented usefulness on compatible cases, without claiming benchmark-wide solver readiness.

Keywords: natural language processing, optimization modeling, knowledge representation, information retrieval, semantic grounding, intelligent information systems

Acknowledgements. The author is deeply grateful to his mother for her continuous emotional support. The author also thanks his PhD advisor, Professor Ioannis Koutis, for his support, guidance, and encouragement.

1 Introduction

1.1 Background and Motivation

Many real-world engineering and operations problems are first described in natural language before they are formalized as optimization models. Analysts, planners, and domain experts often specify resource limits, costs, capacities, proportions, and operational rules in text long before these elements are translated into a mathematical program. This gap between informal problem descriptions and formal optimization artifacts has motivated growing interest in natural-language interfaces for optimization, including assistive auto-formulation systems [16], benchmark efforts such as NL4Opt [17], and more recent large-language-model-based pipelines [3, 8, 23]. The problem is important not only from the perspective of automation, but also for engineering decision support: a system that can reliably connect a textual problem description to an appropriate optimization structure can reduce modeling effort, improve accessibility, and support practitioners during early stages of formalization.

At the same time, full natural-language-to-optimization translation remains difficult. A complete system must determine which optimization template or problem class is appropriate, identify the relevant quantities mentioned in text, align these quantities with the correct model parameters, and ultimately produce a formulation that is structurally and numerically usable. Recent work has shown that large language models and related systems can generate optimization-oriented formulations, solver code, or executable outputs directly from text [3, 4, 23], including via intermediate structured representations such as hierarchical document parsing [18] and broader cross-domain modeling targets [20]. However, such end-to-end settings combine several sources of difficulty at once, making it hard to determine which intermediate capabilities are reliable, which stages contribute most to failure, and which parts of the pipeline already provide practical value for engineering optimization workflows.

This paper adopts a narrower and more transparent perspective. Rather than attempting to generate a complete solver-ready optimization model from text, we focus on an intermediate task that is both practically meaningful and experimentally tractable: given a natural-language optimization description, can a deterministic system retrieve the most compatible optimization schema and then instantiate its scalar parameters from textual numeric evidence? This perspective is motivated by two observations. First, schema retrieval is a necessary prerequisite for many downstream

modeling pipelines because parameter interpretation depends on the underlying problem structure. Second, partial but reliable recovery of scalar parameters can still provide useful modeling support even when full formulation generation remains out of reach. In this sense, retrieval-assisted schema grounding can be viewed as a transparent decision-support layer between free-form language and formal optimization artifacts.

This framing is especially relevant for intelligent decision support. In contrast to opaque end-to-end generation, a retrieval-assisted deterministic pipeline exposes its intermediate decisions: which schema was selected, which scalar slots were considered eligible, which numeric mentions were extracted, and how assignments were made. Such transparency is valuable in optimization settings, where users often need to inspect, verify, and revise system outputs before trusting them in a modeling workflow. Related work on explainable and user-facing optimization systems similarly emphasizes the importance of interpretability, usability, and human-centered support around optimization technology [5, 15]. Our goal is therefore not to replace the modeling expert, but to study whether a simple, reproducible, retrieval-centered system can provide measurable downstream utility for structured optimization problem instantiation.

Guided by this motivation, we study this framework on the NLP4LP benchmark [1, 3]. This restricted formulation is useful precisely because it separates two issues that are often conflated in end-to-end systems: retrieving the correct broad problem structure and grounding the numerical content of the query into the correct scalar roles. The remainder of the introduction positions this perspective relative to prior work, clarifies the scope of the paper, and summarizes the main contributions.

1.2 Related Work

Research at the intersection of natural language processing and optimization has expanded rapidly in recent years. An important early direction has focused on helping users convert textual problem descriptions into formal optimization artifacts. Ramamonjison et al. introduced an assistive auto-formulation framework in which natural-language problem descriptions are mapped to editable optimization-model suggestions, combining automatic support with interactive modeling assistance [16]. This line of work also led to benchmark-oriented efforts such as NL4Opt, which helped establish natural-language-to-optimization as a shared evaluation problem and emphasized the extraction of optimization entities and machine-interpretable structure from text [17]. Related work such as LaTeX2Solver explored hierarchical parsing pipelines for converting semi-structured mathematical documents into optimization-oriented code representations [18]. Together, these studies demonstrate both the importance of the problem and the diversity of interfaces between language and optimization.

A second and increasingly prominent direction uses large language models for end-to-end optimization modeling. OptiMUS studies how large language models, together with solver feedback, can formulate, implement, debug, and solve linear and mixed-integer programming problems from natural-language descriptions [2, 3]. OptLLM likewise proposes an LLM-centered framework augmented with external solvers for converting natural-language optimization requests into formulations, code,

and solutions [23]. CHAIN-OF-EXPERTS takes a multi-agent view of the same problem, coordinating a set of role-specialized LLM agents (e.g., terminology interpretation, modeling, and programming experts) under a conductor that revises the formulation through forward construction and backward reflection [21]. LLMOPT instead fine-tunes a model end-to-end on a five-element optimization formulation together with automated model classification and solver-code generation, targeting broader generalization across problem types [10]. More recently, retrieval-augmented generation has also been explored in this setting. OPT2CODE, for example, combines retrieval, code generation, and judge-based checking to translate linear-programming descriptions into solver-executable code [4]. These systems have advanced the state of the art in end-to-end automation, but they also combine several difficult subproblems at once, including problem understanding, formulation generation, numerical grounding, code production, and downstream validation.

Beyond full model generation, several works have examined intermediate components of the language-to-optimization pipeline. NER4OPT frames optimization modeling support as an information-extraction problem over natural-language descriptions, focusing on optimization-relevant entities and quantities needed for downstream modeling [11]. Recent benchmark work has also broadened the scope beyond linear and mixed-integer programming. For example, TEXT2ZINC introduces a cross-domain dataset for optimization and satisfaction problems in MINIZINC, while CP-BENCH evaluates large language models for constraint modeling across multiple constraint-programming systems [13, 20]. Related work such as EHOP further shows that the linguistic presentation of optimization-style tasks can materially affect model performance, reinforcing the value of studying intermediate language-to-structure capabilities directly [7]. At a broader level, recent surveys and position papers document the rapidly growing interaction between large language models and optimization, including both using optimization to improve language models and using language models to support optimization workflows [8, 9, 22].

Another nearby strand of work concerns explainability and human-centered support for optimization systems. Rather than generating optimization models from text, these approaches use language models to explain schedules, decisions, or optimization outcomes in user-friendly terms [5, 15]. This perspective is relevant to engineering decision support because it emphasizes transparency, trust, and usability in optimization-assisted workflows. However, it addresses a different stage of the pipeline: explaining optimization outputs rather than grounding natural-language problem descriptions into formal model structure.

Our work is most closely related to natural-language-to-optimization modeling, but it occupies a different point in the design space from both end-to-end generation systems and component-level extraction systems. In contrast to approaches that aim to produce complete formulations, executable code, or solver-ready models, we focus on a narrower intermediate task: retrieval-assisted schema grounding followed by scalar parameter instantiation. In contrast to pure extraction approaches, we make downstream slot eligibility depend explicitly on the retrieved schema, so that downstream evaluation remains structurally tied to schema selection quality. This design allows us to isolate an empirically important question that is often obscured in end-to-end

pipelines: whether the main difficulty lies in retrieving the correct broad optimization structure or in grounding the numerical content of the query into the correct scalar roles once that structure has been identified.

The resulting perspective places the paper between full automation and low-level entity extraction. The proposed framework is fully deterministic and transparent, which makes it possible to analyze where gains arise and where residual errors remain. More importantly, it allows us to evaluate a practically meaningful intermediate capability for engineering optimization support without conflating it with full formulation synthesis. We therefore position this paper as a study of retrieval-assisted optimization-model support: not full natural-language-to-optimization compilation, but schema retrieval and schema-conditioned scalar instantiation as measurable and practically useful components of a broader engineering decision-support workflow.

1.3 Problem Scope and Proposed Perspective

The benchmarked task is *schema-conditioned scalar parameter instantiation*: given a natural-language query, the system ranks a fixed catalog of candidate schemas, selects the top-ranked template, and then uses that template to determine which scalar parameter slots are eligible before assigning numeric mentions extracted from the query to those slots by deterministic rules. The scope is deliberately restricted – to scalar parameters, to a fixed schema catalog, and to a fully deterministic pipeline with no learned or large-language-model components, as detailed in Section 2.1. The system is thus neither a complete natural-language-to-optimization compiler nor a purely local entity recognizer, but a transparent intermediate layer intended to quantify how much structured utility can be obtained from deterministic schema retrieval and scalar grounding alone. Strong benchmark performance should accordingly be read as evidence of useful schema grounding and partial parameter recovery, not as complete optimization-model generation.

A defining feature of this formulation is that eligible slots are derived from the *predicted* schema rather than the gold schema, so downstream performance remains retrieval-dependent by construction: if the wrong schema is retrieved, the eligible parameter space may be structurally misaligned even when the query still contains informative numeric evidence. This design lets the experiments measure not only whether the correct template is retrieved but also whether improved retrieval translates into better structured recovery – a distinction that turns out to be central, since schema retrieval on NLP4LP is already strong while the residual difficulty lies downstream (Section 2.4 develops this design in full). The formulation is narrow enough to permit precise evaluation and error analysis, yet still connected to practice: beyond the main benchmark we report restricted downstream validation, including structural-validity analysis and a small solver-backed executable study on compatibility-filtered instances (Section 3.6), which assess downstream relevance without redefining the core task.

1.4 Contributions and Practical Implications

This paper makes four main contributions. First, it contributes a *problem formulation and evaluation framework* that decomposes natural-language optimization understanding into two explicit, separately measurable stages – schema retrieval and schema-conditioned scalar parameter grounding – connected by a retrieval-dependent evaluation design in which eligible parameter slots are determined by the predicted schema rather than the gold schema. Second, it contributes a *transparent methodology*: a fully deterministic pipeline that maps numeric textual evidence to typed optimization-schema parameters through lexical schema retrieval, rule-based numeric extraction, coarse type inference, and no-reuse compatibility-based slot assignment, extended by a lexicon-based optimization-role and admissibility refinement layer that scores slot–mention compatibility using role cues (e.g., capacity, cost, or cardinality-bound language) beyond coarse type agreement; every intermediate decision is inspectable and reproducible. Third, it contributes a *comprehensive empirical diagnosis* of where this pipeline succeeds and fails, combining oracle controls, a strict-metric sensitivity analysis, a numeric-type compatibility ablation, paired bootstrap significance tests, lexical-overlap robustness tests, negative results for three richer deterministic grounding families (global-compatibility, relation-aware, and ambiguity-aware grounding), and an error taxonomy; together these consistently identify semantic quantity-to-role grounding – not schema retrieval – as the dominant bottleneck (an oracle control with perfect schema retrieval improves *Instantiation-Ready* only from 0.5287 to 0.5680, a statistically significant but practically modest gain). Fourth, it contributes *reproducibility and downstream validation*: a public implementation together with restricted structural and solver-backed evidence (60-instance structural, 269-instance executable-attempt, and 20-instance solver-backed subsets) that clearly separates what is reproducible end-to-end from what remains unsupported at benchmark scale.

These contributions also carry a practical implication for intelligent decision support. In many settings the near-term goal is not full automation but tools that help users move from informal descriptions to formal decision models in a controlled and inspectable way. Retrieval-assisted schema grounding serves this goal by narrowing the candidate modeling space, exposing the inferred template structure, and recovering part of the scalar content that a human can then check, correct, or extend – an emphasis on transparency and human oversight consistent with recent work on explainable and user-facing optimization support [5, 15]. The system is therefore not presented as a complete natural-language-to-optimization compiler or as a replacement for an optimization modeler, but as a transparent intermediate layer whose measurable value complements recent large-language-model-based optimization systems by clarifying what a deterministic pipeline can already achieve and where stronger downstream reasoning is still needed [3, 4, 23].

The remainder of the paper is organized as follows. Section 2 presents the problem formulation, retrieval stage, deterministic parameter-instantiation backbone, and evaluation-oriented design choices. Section 3 describes the experimental setup and

reports retrieval performance, downstream utility, structural and solver-backed validation, and error analysis. Section 4 concludes with the main findings, limitations, and directions for future work.

2 Methodology

2.1 Problem Formulation

We study a retrieval-assisted instantiation task for natural-language optimization descriptions. Given a query that describes an optimization problem, the objective is not to generate a complete mathematical program directly. Instead, the system produces two structured outputs: (i) a schema prediction that identifies the most compatible optimization template from a fixed catalog, and (ii) scalar parameter assignments for the retrieved schema. This formulation captures an important intermediate step in natural-language-to-optimization pipelines: before a full model can be constructed, the system must identify an appropriate template and recover the key quantities needed to instantiate it [1, 3, 17].

We cast the task as a two-stage decision process. In the first stage, the system performs *schema retrieval*: given a query q, it ranks a catalog $\mathcal{S} = \{s_1, \ldots, s_M\}$ of candidate optimization schemas and returns the top-ranked schema

$$\hat{s} = \arg\max_{s \in \mathcal{S}} \text{score}(q, s).$$

Each schema represents an optimization template together with its associated parameter structure. In the second stage, the system performs *parameter instantiation*: conditioned on the retrieved schema $\hat{s}$, it extracts numeric mentions from the query and assigns them to the scalar parameter slots specified by $\hat{s}$ using deterministic grounding rules. The second stage is explicitly retrieval-dependent: if schema retrieval is incorrect, the set of eligible parameter slots may also be incorrect, reducing downstream coverage and readiness even when the query still contains informative numeric evidence.

In this paper, the term *schema* refers to the retrieved optimization template together with the scalar parameter structure to be instantiated. The resulting task is narrower than full natural-language-to-optimization translation. The main benchmarked setting does not attempt to synthesize a complete optimization model, recover all variables and constraints, or reconstruct structured parameter objects such as lists, vectors, or dictionaries. Rather, the focus is on whether schema retrieval can anchor a natural-language description to a plausible optimization template, and whether deterministic grounding can recover useful scalar values once that template has been selected.

Our scope is intentionally restricted in three ways. First, benchmark-level downstream evaluation is limited to scalar numeric parameters. Second, the core retrieval-and-grounding pipeline is fully deterministic and uses no large language model, no learned retriever, and no learned extractor. Third, the main benchmark study evaluates schema-conditioned scalar recovery rather than full end-to-end model generation. These restrictions are deliberate: they isolate an interpretable intermediate capability

and allow precise analysis of where performance gains and failures arise. At the same time, the broader experimental section complements this benchmarked formulation with structural and solver-backed subset validation, including structural-validity analysis and a restricted solver-backed study on compatible instances. These additional experiments are intended to assess downstream practical relevance, not to redefine the core task studied here.

Formally, let $G(q)$ denote the set of scalar gold parameters associated with query q, and let $P(\hat{s})$ denote the scalar parameter keys exposed by the retrieved schema $\hat{s}$. The downstream evaluator considers only the eligible slot set

$$E(q, \hat{s}) = G(q) \cap P(\hat{s}), \tag{1}$$

that is, the scalar gold parameters whose keys are present in the predicted schema. This choice makes downstream evaluation retrieval-dependent by construction. Even when a query contains the correct numeric evidence, an incorrect schema can reduce structural overlap and therefore lower downstream performance. The formulation is designed to measure not only whether the correct schema is retrieved, but also whether improved schema retrieval yields measurable gains in structured parameter instantiation.

Retrieval quality is measured by whether the top-ranked schema matches the gold schema, while downstream quality is measured by how well scalar slots from the retrieved schema can be instantiated from the query text. As the experiments later show, retrieval on NLP4LP is already strong, whereas the main remaining difficulty lies in grounding extracted numbers to the correct scalar roles within the chosen schema. The formulation adopted here therefore isolates the stage that currently matters most empirically: deterministic number-to-slot grounding after schema retrieval.

2.2 Retrieval-Based Schema Identification

The first stage of the framework performs *schema identification by retrieval*. Given a natural-language optimization description, the system selects the most compatible candidate from a fixed catalog of benchmark-induced schema documents. We formulate this stage as a ranking problem over a finite candidate set $\mathcal{S}$. Each candidate document is indexed by a schema identifier and paired with the textual schema description used by the evaluation pipeline. Given an input query, the retriever scores the query against every candidate document and returns a ranked list. In the present pipeline, only the highest-ranked candidate is used, so retrieval is treated as a top-1 schema selection problem rather than a shortlist-generation or interactive retrieval task.

It is important to clarify what a "schema candidate" means in this setting. The retrieval corpus is the benchmark-induced catalog used by the evaluation pipeline, which contains 335 candidate schema documents. Thus, the retrieval task in this paper is neither open-domain optimization search nor coarse classification into a small number of broad problem families. Instead, it is a fixed-catalog identification problem: for each query, the system must select the single benchmark schema document that serves as the structural template for downstream grounding. This benchmark-specific

formulation is deliberate because the goal of the paper is to study whether natural-language optimization descriptions can first be anchored to a usable template and then instantiated deterministically.

Operationally, the retrieved schema serves as the structural anchor for downstream instantiation. Through its identifier, the system accesses the associated parameter structure and uses that structure to determine which scalar slots are eligible for filling. The retrieval stage therefore does *not* attempt to reconstruct a full symbolic optimization model, derive solver-ready constraints, or generate variables and objectives directly. Instead, it selects the most plausible template index from which the downstream stage can obtain an expected set of scalar parameter roles [1, 3, 17].

This fixed-catalog setup should also be distinguished from label-only classification. The retriever does not predict an abstract class name from a small closed set of manually engineered categories. Rather, it ranks textual schema documents using their natural-language content. This distinction matters because the empirical question in our benchmark is not merely whether a query belongs to a broad optimization family, but whether it can be aligned to the *specific* schema document whose parameter inventory is then used for downstream slot grounding. Retrieval quality therefore directly affects the structural conditions under which the second stage operates.

We evaluate three deterministic lexical retrieval baselines: BM25 [19], TF–IDF cosine retrieval [14], and latent semantic analysis (LSA) [6]. All three retrievers are deterministic, CPU-based, and operate directly on schema-catalog text without benchmark-specific training, dense neural encoders, or learned reranking. Table 1 summarizes their roles in the pipeline.

For a query q, each retriever produces a score for every candidate schema $s \in \mathcal{S}$, and the predicted schema is

$$\hat{s}(q) = \arg\max_{s \in \mathcal{S}} \operatorname{score}(q, s). \tag{2}$$

Only $\hat{s}(q)$ is passed to the downstream instantiation stage. This top-1 design matches the evaluation goal of the paper: we ask whether the system can identify a single usable schema that supports deterministic slot filling, rather than whether it can produce a longer candidate list for later interaction or reranking.

To interpret retrieval performance, we use two diagnostic references in the broader pipeline. The first is an *oracle* control, which replaces the retrieval output with the gold schema identifier and therefore exposes downstream behavior under perfect schema selection. The second is a *random* reference. For retrieval-only evaluation, this random reference is the theoretical top-1 chance rate under uniform selection over the 335 candidate schema documents, namely $1/335 \approx 0.0030$. For downstream evaluation, "random" instead denotes a uniformly chosen schema passed through the same deterministic instantiation pipeline. These references are included only to calibrate interpretation; they are not competing learned retrievers.

Because the retrieved schema determines the candidate slot inventory used downstream, retrieval errors propagate structurally. If the wrong schema is selected, the second stage may inherit a slot set that only partially overlaps the gold instance or that mismatches the intended scalar roles altogether. This can reduce downstream

Table 1 Deterministic retrieval backbones used for schema identification.

Method	Representation	Role in pipeline
BM25	Probabilistic lexical weighting	Sparse lexical baseline
TF–IDF	Weighted bag-of-words cosine similarity	Main deterministic retrieval backbone
LSA	Latent projection of TF–IDF features	Lower-dimensional lexical-semantic baseline

coverage, type agreement, and per-query readiness even when the query still contains relevant numeric evidence. Conversely, strong retrieval creates more favorable structural conditions for deterministic grounding by supplying a more appropriate set of candidate slots.

This formulation emphasizes retrieval as *schema grounding* rather than full optimization-model reconstruction: the first stage maps a free-form optimization description to a single candidate template whose scalar parameter structure can then be instantiated transparently. Retrieval is an important enabling step in the pipeline, but, as the experiments show, not the dominant empirical bottleneck.

2.3 Deterministic Parameter Instantiation

Given a natural-language query and a predicted schema, the second stage of the framework instantiates scalar parameter slots from textual numeric evidence. The output of this stage is a set of slot–value assignments associated with the retrieved schema. In the main benchmarked setting, this stage does not generate a complete optimization model and does not recover variables, constraints, or structured parameter objects. Its purpose is narrower: to recover scalar values that can be aligned with the parameter structure of the retrieved template.

The instantiation stage begins by extracting candidate numeric mentions from the query text using a fixed rule-based pattern matcher. The extractor recognizes integers, decimals, percentage expressions, currency-like forms, and comma-separated magnitudes. In degraded query settings, masked placeholders are preserved as special tokens but treated as numerically unknown, so they cannot support exact value recovery. Each extracted mention is then assigned a coarse type label from four buckets: *percent*, *integer*, *currency*, and *float*. Mention typing relies on deterministic surface cues such as percentage markers, currency markers, integer-valued forms, and local lexical context.

The predicted schema determines which parameter slots are eligible for instantiation. Operationally, the system obtains the parameter keys associated with the retrieved template and then restricts attention to scalar-valued slots. This restriction is deliberate. List-valued and dictionary-valued parameters correspond to more structured objects whose recovery would require additional representational assumptions, so the present framework focuses on scalar numeric instantiation. The instantiation

problem is therefore defined over a finite set of eligible scalar slots together with a pool of extracted numeric candidates.

For each eligible slot, an expected type is inferred from its parameter name using deterministic rules. Parameters whose names indicate rates or fractions are mapped to *percent*; names indicating counts or cardinalities are mapped to *integer*; names suggesting monetary quantities are mapped to *currency*; and all remaining scalar slots are mapped to *float*. This shared typed representation places slots and extracted numeric mentions in a common compatibility space before assignment.

The simplest assignment rule used in the framework is typed greedy matching. Let $\mathcal{P} = \{p_1, \ldots, p_m\}$ denote the eligible scalar slots induced by the predicted schema, and let $\mathcal{T} = \{t_1, \ldots, t_n\}$ denote the extracted numeric mentions. In the typed-greedy setting, the selected mention for slot p_i is the highest-scoring unused candidate under a deterministic compatibility rule,

$$t_i^* = \arg \max_{t \in \mathcal{T} \setminus \mathcal{U}_{i-1}} \text{compat}(p_i, t), \tag{3}$$

where $\mathcal{U}_{i-1}$ is the set of mentions already assigned in previous steps. The compatibility function prioritizes agreement between the expected slot type and the inferred mention type, together with deterministic tie-breaking based on local cues and extraction order. Once a mention is assigned, it is removed from the candidate pool and cannot be reused for later slots. This yields a one-pass deterministic assignment rule rather than a global combinatorial optimizer.

The broader method family evaluated in the paper extends this same backbone. In addition to lexical typed-greedy baselines, we consider deterministic grounding variants that modify how slot–mention compatibility is scored or revised after an initial assignment. These include constrained matching, semantic repair, optimization-role repair, acceptance reranking, hierarchical acceptance reranking, and relation-aware, global-compatibility, and ambiguity-aware variants. Although these methods differ in scoring or repair strategy, they share the same core structure: deterministic numeric extraction, schema-conditioned scalar slot selection, coarse type inference, and no-reuse slot filling. This common backbone is summarized in Algorithm 1 and illustrated in Figure 1.

We use *optimization-role compatibility* to refer to a deterministic, lexicon-based refinement of the basic type-compatibility signal used by `tfidf_optimization_role_repair` and the newer relation-aware and global-compatibility variants. Beyond the coarse type label (*percent*, *integer*, *currency*, *float*), each slot name and its local query context are matched against a fixed lexicon of optimization-role cue words (e.g., terms indicating a capacity limit, a cost coefficient, a cardinality bound, or a derived count). When a candidate mention's local context shares a role cue with the slot, the deterministic assignment score in Eq. (3) receives an additive bonus; when the mention's inferred type is incompatible with the slot's expected type, it receives a penalty large enough to make the mention non-preferred (though not strictly inadmissible, since ties and missing alternatives can still cause an incompatible mention to be assigned). We use *admissibility* in this paper only in this graded, score-based sense – a slot–mention pairing is more or less preferred

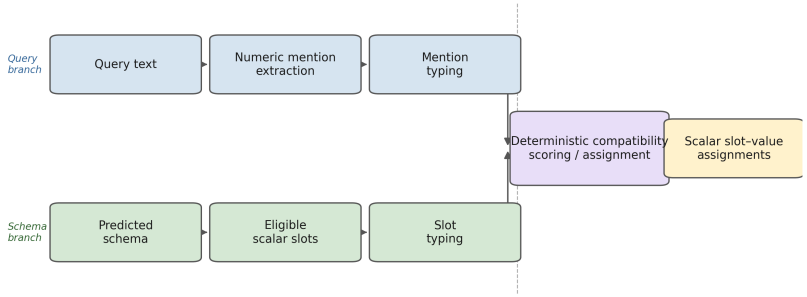


Fig. 1 Schema-conditioned deterministic parameter instantiation. Numeric mentions are extracted from the query, typed using surface cues, filtered through the scalar slot inventory of the predicted schema, and assigned by a no-reuse compatibility rule. Downstream variants differ mainly in how slot–mention compatibility is scored or repaired.

under the compatibility rule – rather than as a hard combinatorial constraint verified against a formal model of the optimization problem; the framework does not currently guarantee that an admissible-by-type assignment is also feasible or optimal for the underlying (unobserved) mathematical program. This is a heuristic, rule-based layer, and we report it as such: it is designed to break ties among type-compatible candidates using domain-typical lexical cues, not to provide a formal correctness guarantee.

To isolate the effect of type-aware matching, we also evaluate a *numeric-type compatibility ablation*. Earlier compatibility logic mishandled *float*-valued slots, underestimating their compatibility with extracted numeric mentions; the corrected logic resolves this. As the experiments later show, this correction materially improves *TypeMatch* and *InstantiationReady* while leaving extraction-dependent quantities much less changed, confirming that the main downstream gains arise from improved number-to-slot grounding rather than from changes in retrieval or mention extraction. We explain the correction only once here and refer to the comparison throughout as the numeric-type compatibility ablation.

In short, the instantiation stage is intentionally simple: numeric evidence is extracted from the query, typed using deterministic cues, filtered through the scalar parameter structure of the predicted schema, and assigned to slots by a deterministic no-reuse procedure. The resulting slot–value mapping is then evaluated downstream. In the broader paper, this benchmarked scalar-instantiation stage is complemented by restricted downstream validation on subsets, but the core methodology here remains a deterministic, schema-conditioned scalar grounding pipeline.

2.4 Evaluation-Oriented Design Choices

The methodology is intentionally designed so that downstream evaluation remains sensitive to schema retrieval quality. In particular, the set of parameter slots eligible for instantiation is determined from the *predicted* schema rather than from the gold schema. This choice is central to the formulation. If eligible slots were defined directly

Algorithm 1 Deterministic parameter instantiation pipeline. Numeric mentions are extracted from the query and typed using surface cues. In parallel, the predicted schema provides the eligible scalar slots, whose expected types are inferred from slot names. A deterministic compatibility-based assignment procedure then maps numeric mentions to slots without token reuse. Different downstream variants share this backbone but differ in how slot–mention compatibility is scored or repaired.

Require: Query text q, predicted schema $\hat{s}$
Ensure: Slot–value mapping A
Extract numeric mentions $\mathcal{T}$ from q
Infer a coarse type for each mention in $\mathcal{T}$
Obtain parameter keys from schema $\hat{s}$
Restrict to eligible scalar slots $\mathcal{P} = \{p_1, \ldots, p_m\}$
Infer an expected type for each slot in $\mathcal{P}$
Initialize assignments $A \leftarrow \emptyset$ and used-mention set $\mathcal{U} \leftarrow \emptyset$
for each slot $p_i \in \mathcal{P}$ **do**
    Define candidate set $\mathcal{C}_i \leftarrow \mathcal{T} \setminus \mathcal{U}$
    if $\mathcal{C}_i = \emptyset$ **then**
        continue
    end if
    Score mentions in $\mathcal{C}_i$ using a deterministic slot–mention compatibility rule
    Select the highest-ranked mention t_i^*
    Assign $A[p_i] \leftarrow \text{value}(t_i^*)$
    Update $\mathcal{U} \leftarrow \mathcal{U} \cup \{t_i^*\}$
end for
return A

from the gold record, downstream coverage and readiness would no longer reflect schema-selection quality, and the evaluation would partially decouple retrieval from instantiation. By conditioning slot eligibility on the retrieved template, the framework ensures that retrieval errors propagate naturally into downstream performance through reduced structural overlap, lower slot availability, and weaker query-level readiness. At the same time, the evaluation avoids leakage: gold information is used for scoring, not for determining which schema is instantiated at inference time.

A second design choice restricts the main benchmarked downstream evaluation to scalar numeric parameters, which provide a common and reproducible substrate for comparing retrieval-conditioned instantiation across heterogeneous templates while avoiding the additional alignment and dimensionality assumptions required to reconstruct list- or dictionary-valued objects (Section 2.1).

As introduced in Section 2.2, the oracle and random references serve the same interpretive purpose throughout the paper. We refer to the oracle condition as an *oracle control* (or oracle reference) rather than a formal upper bound: because *InstantiationReady* depends on per-query type-inference and slot-eligibility details that are not guaranteed to be monotonic in schema correctness, the oracle condition is not mathematically certain to dominate every non-oracle method on every metric or subset, and Section 3.6 reports a concrete case (the 20-instance solver-backed subset) where it does not.

The core study is intentionally solver-free and fully deterministic: no feasibility test, optimization run, or objective-level verification enters the core metrics, and the pipeline uses no large language model, learned retriever, or learned extractor. This keeps the central evaluation transparent and reproducible and lets the experiments answer a focused question – how much structured instantiation utility deterministic retrieval and rule-based grounding can provide – while downstream executability is examined separately on the restricted subsets of Section 3.6.

These choices also determine how the metrics should be read. *InstantiationReady* is a diagnostic readiness criterion based on coverage and type consistency, not a guarantee of model correctness or solver compatibility; strong schema retrieval is not full optimization-model recovery; and successful scalar assignment is not feasibility- or optimality-verified instantiation. The framework is best viewed as a retrieval-assisted deterministic instantiation system that quantifies how schema grounding affects structured parameter recovery, with the restricted subset analyses addressing the separate question of downstream executable behavior on compatible cases.

Note on generative-AI assistance. The retrieval-and-grounding pipeline studied in this paper is itself fully deterministic and uses no large language model, learned retriever, or learned extractor (see above). Separately, during preparation of this work the author used ChatGPT and Gemini to assist with aspects of manuscript writing and used Cursor and GitHub Copilot to assist with coding; the author reviewed, verified, and edited all generated content and takes full responsibility for the content of the publication. No generative-AI tool is listed as an author, consistent with Springer Nature's authorship policy.

3 Experiments

3.1 Experimental Setup

We evaluate the proposed retrieval-assisted instantiation pipeline on the NLP4LP benchmark [1, 3], a curated collection of linear- and mixed-integer-programming problem descriptions in which each instance pairs a natural-language description with a gold parameter record, a reference solver implementation, and, where available, a reference solution. Unless otherwise stated, the main benchmark results are measured on the benchmark test split, which contains 331 query instances. Each test query has exactly one gold optimization schema. Each test query is ranked against the benchmark-induced schema catalog used by the evaluation pipeline, which contains 335 candidate schema documents. Accordingly, retrieval is a 335-way top-1 identification task, and the corresponding random top-1 reference is the theoretical chance rate under uniform selection over this catalog, namely $1/335 \approx 0.0030$.

To assess robustness to surface degradation and reduced context, we evaluate three benchmark query variants. The *orig* variant uses the original problem descriptions. The *noisy* variant applies controlled lexical corruption, including lowercasing, token deletion, stopword reduction, and masking of some numeric expressions. The *short* variant retains only the first sentence of each description. These variants allow us to test the same retrieval-and-grounding pipeline under full context, degraded wording, and substantially reduced context.

The retrieval stage ranks candidate schemas using only schema-catalog text. In the final reported benchmark comparison, we evaluate three deterministic lexical retrieval baselines: BM25 [19], TF–IDF cosine retrieval implemented with SCIKIT-LEARN [14], and latent semantic analysis (LSA), obtained by applying truncated singular value decomposition to TF–IDF vectors [6]. In all downstream experiments, only the top-ranked schema is passed to the instantiation stage. We also report an *oracle* control in which the retrieved schema is replaced by the gold schema, thereby isolating downstream grounding difficulty from retrieval error.

Given a query and a predicted schema, the downstream stage instantiates scalar parameter slots deterministically from the query text. Numeric mentions are extracted using a fixed rule-based pattern matcher. In the *noisy* variant, masked numeric placeholders are preserved as special tokens but treated as numerically unknown; they may therefore still indicate the presence of a quantity without supporting exact value recovery. We restrict the core benchmarked downstream evaluation to scalar numeric parameters and exclude list-valued and dictionary-valued parameters so that the benchmark targets scalar slot grounding rather than structured arrays, matrices, or other higher-order objects.

The downstream candidate slot set is derived from the *predicted* schema rather than the gold schema. When schema metadata explicitly provides parameter keys, we use those keys; otherwise, we extract keys from the predicted parameter record. These predicted keys are then intersected with scalar gold parameters to determine the subset of slots eligible for evaluation. This makes downstream evaluation intentionally retrieval-dependent: if the wrong schema is retrieved, the candidate slot inventory may already be structurally misaligned with the gold instance, which can reduce both attainable coverage and value accuracy even when the query still contains relevant numbers.

To assign extracted numbers to slots, we evaluate multiple deterministic grounding families that differ in how much structural guidance they exploit. Across all methods, scalar slots are typed using four lightweight rule-based categories: *percent*, *integer*, *currency*, and *float*. Extracted numeric mentions are mapped into the same categories using surface cues such as percentage markers, currency symbols, and integer-valued forms. The benchmark includes the original typed-greedy baselines based on TF–IDF, BM25, and LSA; an oracle typed-greedy control; and five previously reported TF–IDF-based deterministic grounding variants with additional constraints or repair mechanisms, namely constrained matching, semantic repair, optimization-role repair, acceptance reranking, and hierarchical acceptance reranking. To broaden the comparison set, we further evaluate three additional method families: *global compatibility grounding*, *relation-aware linking*, and *ambiguity-aware grounding*. This expanded design allows us to compare simple deterministic assignment against richer structured grounding strategies while keeping the overall pipeline transparent and reproducible.

We report both retrieval effectiveness and downstream instantiation quality. *Schema_R@1* is the fraction of test queries whose top-ranked schema matches the gold schema. For downstream evaluation, let G_i denote the scalar gold slots for query i that are eligible for evaluation after intersecting gold scalar parameters with the candidate slots induced by the predicted schema, and let $A_i \subseteq G_i$ denote the subset of

those slots that receive an assignment. Then *Coverage* is

$$\mathit{Coverage} = \frac{\sum_i |A_i|}{\sum_i |G_i|}, \tag{4}$$

that is, the fraction of eligible scalar gold slots that receive any assignment. *TypeMatch* is the fraction of assigned slots whose predicted numeric type matches the expected slot type,

$$\mathit{TypeMatch} = \frac{\sum_i |\{s \in A_i : \mathrm{type}(\hat{y}_{i,s}) = \mathrm{type}(y_{i,s})\}|}{\sum_i |A_i|}, \tag{5}$$

where $\hat{y}_{i,s}$ and $y_{i,s}$ denote the predicted and gold values for slot s of query i.

To measure value accuracy, we report *Exact20_on_hits*. This metric is computed only on the subset of *schema-hit* queries for which a comparable numeric prediction and gold value are both available for the slot under consideration. A slot is counted as correct if the predicted numeric value falls within a relative error tolerance of 20% of the gold value. Because meaningful numeric comparison requires both schema correctness and a comparable numeric pair, *Exact20_on_hits* is intentionally normalized on this eligible subset rather than on all test queries. It should therefore be interpreted as a conditional value-grounding measure rather than as an end-to-end benchmark score.

Our main end-to-end utility metric is *InstantiationReady*. For query i, let $\mathit{Coverage}_i$ and $\mathit{TypeMatch}_i$ denote the per-query coverage and type-match rates (Eqs. (4)–(5)) computed over the predicted-schema-conditioned eligible slot set $E(q_i, \hat{s}_i)$ (Eq. (1)). A query is counted as *InstantiationReady* if both rates reach a fixed threshold,

$$\mathit{InstantiationReady}_i = \mathbb{1}[\mathit{Coverage}_i \geq 0.8 \ \wedge \ \mathit{TypeMatch}_i \geq 0.8], \tag{6}$$

and the reported *InstantiationReady* score is the mean of $\mathit{InstantiationReady}_i$ over the benchmark denominator. Because $E(q_i, \hat{s}_i)$ is itself schema-dependent, an incorrect schema prediction typically depresses $\mathit{Coverage}_i$ and lowers the chance of reaching the threshold, but exact schema match is not enforced as a separate hard gate in this rule. Thus, *InstantiationReady* is a per-instance usability-threshold criterion that combines schema-conditioned slot coverage and type compatibility into a single diagnostic, rather than an all-or-nothing full-coverage criterion.

Unless otherwise stated, the retrieval metrics and the main downstream end-to-end metrics are reported over the full benchmark denominator for each variant. The only exception is *Exact20_on_hits*, which is explicitly conditional on the subset of schema-hit cases with comparable numeric values. We additionally report bootstrap confidence intervals and selected paired significance analyses for key retrieval and downstream method pairs, and overlap-based stress tests – including stratified retrieval performance by lexical-overlap level and retrieval ablations under sanitized text conditions such as number removal and stopword stripping – to test whether strong retrieval may partly reflect benchmark-specific lexical overlap (Section 3.5).

To isolate the effect of corrected numeric-type compatibility, we also report a numeric-type compatibility ablation on four representative downstream methods across all three query variants, comparing downstream metrics before and after the correction described in Section 2.3. This ablation is especially relevant for *float*-valued

Table 2 Summary of the experimental blocks used in the paper.

Block	Primary purpose	Main outputs
Main benchmark	Retrieval and scalar grounding on NLP4LP	Schema_R@1, Coverage, TypeMatch, Exact20_on_hits, InstantiationReady
Robustness analyses	Sensitivity to degraded input and lexical overlap	Orig / noisy / short comparisons, overlap-based stress tests
Numeric-type-compatibility ablation	Isolate effect of corrected type handling	TypeMatch and InstantiationReady changes
Significance testing	Assess whether reported gaps are statistically robust	Paired bootstrap p-values and confidence intervals
Engineering subset validation (60 inst.)	Assess downstream structural usefulness	Structural-validity and instantiation-completeness rates
Executable-attempt subset (269 inst.)	Document restricted downstream executability and blockers	Executable, solver-success, feasibility, and objective-produced rates
Final solver-backed subset (20 inst.)	Real, restricted solver execution via SciPy HiGHS shim	Executable, solver-success, feasible, and objective-produced rates

slots. Both the main benchmark and the ablation are measured on the canonical NLP4LP test benchmark.

Finally, to complement the core benchmark with downstream practical evidence, we include three structural and solver-backed validation studies (Section 3.6): (i) a structural subset experiment, (ii) an executable-attempt study on the largest deterministic executable-eligible subset available in the local cache, and (iii) a restricted solver-backed final-attempt study on a small compatibility-filtered subset. These auxiliary studies are not replacements for the main benchmark; rather, they assess whether the recovered structure can support increasingly concrete downstream optimization behavior on restricted subsets. Table 2 summarizes the overall experimental design.

All experiments are deterministic and primarily CPU-based. The implementation uses standard data-processing and retrieval libraries, including Hugging Face `datasets` [12] and SCIKIT-LEARN [14]. The core benchmarked evaluation uses no large language model prompting, no custom neural training procedure, and no solver-based scoring.

3.2 Schema Retrieval Performance

We first evaluate schema retrieval in isolation to determine whether selecting the correct benchmark schema is itself a major source of error in the present fixed-catalog setting. Table 3 reports *Schema_R@1*, defined as the fraction of test queries for which the top-ranked retrieved schema matches the gold schema, across the three query variants described in Section 3.1. Retrieval is evaluated against the benchmark-induced

catalog of 335 candidate schema documents. Accordingly, the random reference corresponds to the theoretical top-1 chance rate under uniform selection over this catalog, namely $1/335 \approx 0.0030$.

The purpose of this experiment is deliberately limited. We do not treat it as an open-domain optimization retrieval task or as evidence that lexical retrieval alone would suffice in broader real-world schema libraries. Rather, within the benchmark-induced candidate set, we ask whether simple deterministic retrieval can identify the single schema document that provides the downstream scalar slot inventory used by the grounding stage. This benchmark-scoped retrieval analysis is therefore intended to characterize the structural conditions under which downstream instantiation operates, not to claim a general solution to schema retrieval in unrestricted optimization-modeling environments.

Within this fixed-catalog benchmark, lexical schema retrieval is already strong. On the original queries, TF–IDF achieves the best performance with *Schema_R@1* = 0.9094, followed by BM25 at 0.8822 and LSA at 0.8459. All three methods are far above the random baseline, indicating that the benchmark descriptions contain enough schema-level textual evidence for recovering the correct benchmark template in most cases. Thus, in the canonical test setting studied here, schema identification is substantially easier than the downstream instantiation problem studied later in the paper.

Retrieval also remains strong under controlled lexical degradation. On the *noisy* queries, TF–IDF, BM25, and LSA achieve *Schema_R@1* values of 0.9033, 0.8912, and 0.8882, respectively. These values are close to their corresponding results on the original queries. This stability suggests that benchmark-level schema identification is not highly sensitive to moderate perturbations such as lowercasing, token deletion, stopword reduction, and numeric masking, even though the retrieval backbones themselves remain lexical rather than semantics-aware.

The *short* setting is more challenging. When each description is truncated to its first sentence, *Schema_R@1* decreases to 0.7795 for TF–IDF, 0.7674 for BM25, and 0.7644 for LSA. This drop is expected because truncation removes contextual details that often distinguish closely related benchmark formulations. Even so, performance remains well above chance for all three methods, showing that substantial benchmark-specific schema evidence is often present in the opening sentence alone.

Across all three variants, TF–IDF again achieves the best average retrieval accuracy (0.8641), with BM25 close behind (0.8469) and LSA somewhat lower but still competitive (0.8328). We therefore use TF–IDF as the main retrieval backbone in the downstream experiments. More importantly, the absolute retrieval levels in Table 3 establish the main benchmark-level interpretive point for the remainder of the paper: within this fixed-catalog NLP4LP setting, the dominant limitation is not selecting a plausible schema document, but grounding the numerical content of the query to the appropriate scalar roles once such a schema has been selected. This conclusion is intentionally benchmark-scoped and should not be read as a claim that schema retrieval is solved in broader optimization-modeling settings.

Table 3 Schema retrieval accuracy across query variants, reported as *Schema_R@1*. Each value is computed over all 331 test queries in the corresponding variant. The random baseline is the theoretical top-1 chance rate under uniform schema selection over the 335 candidate schemas, i.e., $1/335 \approx 0.0030$. Best result in each column is shown in bold.

Baseline	Orig	Noisy	Short	Avg.
Random	0.0030	0.0030	0.0030	0.0030
LSA	0.8459	0.8882	0.7644	0.8328
BM25	0.8822	0.8912	0.7674	0.8469
TF–IDF	**0.9094**	**0.9033**	**0.7795**	**0.8641**

3.3 Downstream Utility for Problem Instantiation

We next evaluate whether correct or near-correct schema retrieval translates into useful downstream instantiation within the restricted scalar-grounding setting studied in this paper. In the proposed pipeline, the retrieved schema determines the candidate scalar slot inventory, after which deterministic numeric extraction and slot assignment are applied as described in Section 3.1. The central question is not whether the system can recover a complete optimization model, but whether successful schema selection yields scalar slot–value assignments that are sufficiently complete and type-consistent to provide measurable intermediate support for optimization modeling.

Table 4 reports the main downstream results on the *orig* queries for the original deterministic grounding family. Three findings are most important. First, deterministic retrieval-assisted grounding recovers a substantial amount of scalar structure on the original benchmark inputs. Under the strongest non-oracle setting among the original methods, `tfidf_typed_greedy` reaches *Coverage* = 0.8609, *TypeMatch* = 0.7453, and *InstantiationReady* = 0.5287. The corresponding BM25 and LSA typed-greedy variants remain close, with *InstantiationReady* values of 0.5196 and 0.5076, respectively. Within this benchmarked scalar task, these results show that a deterministic retrieval-assisted pipeline can recover a large fraction of eligible scalar assignments with the correct coarse type.

Second, the best conditional value-accuracy score does not coincide with the best end-to-end utility. `tfidf_constrained` achieves the strongest *Exact20_on_hits* score (0.3239), and `tfidf_semantic_ir_repair` achieves the strongest non-oracle *TypeMatch* score (0.7529), but neither yields the best *InstantiationReady*. As defined in Section 3.1, *Exact20_on_hits* is a conditional metric computed only on eligible schema-hit cases with comparable numeric values, whereas *InstantiationReady* is a threshold-based, full-denominator query-level criterion. The results therefore show that local gains on a restricted comparable subset do not automatically translate into stronger query-level readiness. In this benchmark, downstream usefulness depends on balancing coverage, type compatibility, and retrieval-conditioned structural fit rather than maximizing a single local matching score.

Third, the oracle control confirms that retrieval matters but is not the dominant remaining source of error within this benchmark. Replacing `tfidf_typed_greedy` with `oracle_typed_greedy` increases *Coverage* from 0.8609 to 0.9151, *TypeMatch* from 0.7453 to 0.7998, and *InstantiationReady* from 0.5287 to 0.5680. These gains are meaningful but still modest relative to the remaining gap to perfect instantiation. Even when the correct schema is supplied, substantial difficulty remains in deciding which extracted quantity should fill which scalar slot. This is the central benchmark-level empirical point of the paper: on NLP4LP, schema retrieval is already comparatively strong, whereas downstream number-to-slot grounding remains the main unresolved source of error.

To test whether richer deterministic grounding can materially improve this stage, we evaluate three additional downstream method families: global compatibility grounding (GCG), relation-aware linking (RAL), and ambiguity-aware grounding (AAG), spanning eleven variants across the three query forms. The most important outcome is negative but informative: none of these families surpasses the simple `tfidf_typed_greedy` baseline on the original benchmark (Table 8 in Section 3.4 reports the representative results with per-method deltas). The best representative, `tfidf_relation_aware_basic`, reaches *InstantiationReady* = 0.4985, still below the typed-greedy baseline. The strongest tested global-compatibility and ambiguity-aware representatives, `tfidf_global_compat_full` and `tfidf_ambiguity_aware_beam`, perform substantially worse at 0.4320 and 0.4230, respectively, while the abstention-heavy ambiguity variant becomes overly conservative (0.0272). These results therefore strengthen the paper's main conclusion: the bottleneck is not merely the absence of additional deterministic structure or more careful search. Rather, the underlying quantity-to-role grounding problem remains difficult even after adding richer assignment families.

A related observation is that several of the strongest original non-oracle methods perform very similarly on *orig*. The gap among TFIDF-TG (0.5287), TFIDF-AR (0.5257), BM25-TG (0.5196), and TFIDF-HAR (0.5196) is small. The additional families do not change this picture. Instead, they suggest that once schema retrieval is already strong, incremental changes to local assignment, compatibility scoring, beam search, or conservative abstention produce only limited gains unless they resolve the deeper semantic ambiguity of quantity-to-role matching. Put differently, the main downstream opportunity is not simply choosing among nearby deterministic search strategies, but improving how extracted numbers are semantically anchored to the correct parameter roles.

Table 5 summarizes representative methods across the three query variants. The cross-variant comparison helps separate two different failure modes. In the *noisy* variant, retrieval remains fairly strong, but downstream readiness collapses because many numeric values are intentionally masked; exact value recovery is therefore structurally underdetermined. This is reflected in very low *InstantiationReady* values across all methods, including the oracle control. In the *short* variant, the main issue is not masking but missing context: the first sentence often does not contain enough quantitative or semantic evidence to support broad slot filling. Here the main bottleneck becomes low *Coverage*. Among the representative original methods, `tfidf_acceptance_rerank`

Table 4 Main downstream results on the *orig* queries for the original deterministic grounding family. *Coverage*, *TypeMatch*, and *InstantiationReady* use the full 331-query denominator. *Exact20_on_hits* is conditional on eligible schema-hit cases with comparable numeric values. Best non-oracle result in each column is bold.

Method	Coverage	TypeMatch	Exact20	InstReady
TFIDF-TG	**0.8609**	0.7453	0.1834	**0.5287**
BM25-TG	0.8509	0.7336	0.1884	0.5196
LSA-TG	0.8267	0.7054	0.1822	0.5076
Oracle-TG	0.9151	0.7998	0.1745	0.5680
TFIDF-CON	0.8082	0.7093	**0.3239**	0.4230
TFIDF-SIR	0.7787	**0.7529**	0.2734	0.4864
TFIDF-ORR	0.8218	0.7016	0.3036	0.4411
TFIDF-AR	0.8302	0.7261	0.1768	0.5257
TFIDF-HAR	0.8121	0.7097	0.1771	0.5196

Abbreviations: TFIDF-TG = `tfidf_typed_greedy`, BM25-TG = `bm25_typed_greedy`, LSA-TG = `lsa_typed_greedy`, Oracle-TG = `oracle_typed_greedy`, TFIDF-CON = `tfidf_constrained`, TFIDF-SIR = `tfidf_semantic_ir_repair`, TFIDF-ORR = `tfidf_optimization_role_repair`, TFIDF-AR = `tfidf_acceptance_rerank`, and TFIDF-HAR = `tfidf_hierarchical_acceptance_rerank`.

achieves the strongest *InstantiationReady* on *short* (0.0211), but the absolute values remain very low, indicating that aggressive truncation removes much of the information needed for reliable scalar grounding.

Taken together, these results support a narrower but more defensible interpretation of system utility: on original benchmark inputs, deterministic retrieval-assisted grounding produces partially usable scalar instantiations for a substantial fraction of cases, and adding richer deterministic structure does not close the remaining gap to the oracle control. These downstream results are meaningful as benchmarked evidence about this restricted intermediate task, not as a claim of full semantic understanding or broad real-world optimization-model recovery.

3.4 Error Analysis and Ablation Studies

To better understand the remaining gap between strong schema retrieval and reliable downstream instantiation, we examine three complementary diagnostics: a diagnostic error taxonomy from the main benchmark, the numeric-type compatibility ablation, and a comparison against the three additional downstream method families. Together, these analyses clarify where the deterministic pipeline still fails.

A first point is that the dominant residual errors arise primarily *after* schema retrieval rather than *at* schema retrieval. As shown in Section 3.2, TF–IDF retrieves the correct schema for 0.9094 of the *orig* test queries, leaving limited room for large gains from retrieval alone. The diagnostic breakdown in Table 6 is consistent with this interpretation. Although wrong-schema retrieval remains a real source of failure,

Table 5 Cross-variant downstream summary for representative original methods. The *noisy* variant should be interpreted cautiously because masked numeric values make exact grounding structurally difficult even when the schema is correct.

Method	O-Cov	N-Cov	S-Cov	O-IR	N-IR	S-IR
TFIDF-TG	0.8609	0.7697	0.1065	0.5287	0.0393	0.0151
BM25-TG	0.8509	0.7650	0.1062	0.5196	0.0423	0.0151
TFIDF-AR	0.8302	0.7315	0.1145	0.5257	0.0423	**0.0211**
Oracle-TG	0.9151	0.8196	0.1258	**0.5680**	0.0423	0.0151

O = *orig*, N = *noisy*, S = *short*, and IR = *InstantiationReady*.

larger diagnostic counts are associated with downstream grounding effects, especially type mismatches, role confusion among semantically related quantities, and wrong-slot disambiguation. Thus, once the structural template is approximately correct, the main difficulty becomes deciding which extracted number should populate which scalar role.

Among these downstream phenomena, type-related failures are especially influential. In the benchmark, many scalar values are not explicitly marked as percentages or currency amounts and are not naturally restricted to integer form. These more weakly marked continuous quantities are therefore more vulnerable to type-handling and compatibility errors than clearly signaled surface forms. Before correction, such cases disproportionately suppressed both *TypeMatch* and query-level *InstantiationReady*. This matters substantively, not merely implementation-wise: type compatibility is one of the gates through which a plausible extracted quantity becomes a usable slot filling. The corrected benchmark therefore changes the interpretation of the system. The pipeline is no longer best described as failing mainly because it cannot identify the right schema; rather, it identifies the schema correctly in most cases and then encounters its main remaining difficulty in semantically grounded number-to-slot assignment.

The error taxonomy should be interpreted as diagnostic rather than as a disjoint exhaustive partition. The categories in Table 6 are approximate counts derived from the final measured benchmark and associated diagnostic reports, and a single query may contribute to more than one category. We use them to identify the main error families, not to claim a formal one-label decomposition of all failures. Even with that caution, the pattern is clear: the largest counts are downstream and semantics-related, whereas unsupported schemas are absent and pure extraction misses are comparatively rare.

Table 7 quantifies the numeric-type compatibility ablation. On the *orig* queries, the corrected logic improves *TypeMatch* by 0.4956 for TFIDF-TG, 0.4300 for TFIDF-ORR, 0.4619 for TFIDF-HAR, and 0.5221 for Oracle-TG. These are large changes, and they produce correspondingly large gains in *InstantiationReady*. For example, TFIDF-TG rises from 0.0695 to 0.5287, and Oracle-TG rises from 0.0785 to 0.5680. By contrast, extraction-dependent quantities change much less, exactly as expected from the nature of the correction: the system extracts similar candidate numbers, but now handles value-type compatibility with the target slot much more accurately.

The three additional grounding families provide a stronger test of whether richer downstream structure can resolve the remaining errors—global compatibility grounding (GCG), relation-aware linking (RAL), and ambiguity-aware grounding (AAG), across eleven variants and the three benchmark query forms. The most informative result is that they do *not* overturn the bottleneck diagnosis. As summarized in Table 8, none of the strongest new representatives surpasses the original `tfidf_typed_greedy` baseline on *orig*. The best new representative, `tfidf_relation_aware_basic`, reaches *InstantiationReady* = 0.4985, still below TFIDF-TG (0.5287). The strongest tested global-compatibility and ambiguity-aware representatives, `tfidf_global_compat_full` and `tfidf_ambiguity_aware_beam`, fall further to 0.4320 and 0.4230, respectively, and the abstention-heavy ambiguity variant becomes overly conservative (0.0272). Thus, even after adding more structured local linking, slot–slot compatibility terms, beam-style search, and explicit ambiguity handling, the remaining grounding errors are not easily removed.

This negative result is important because it shows that the residual gap is not simply due to a lack of downstream method variety. Rather, the unresolved failures reflect genuinely difficult semantic ambiguities: totals versus unit coefficients, lower versus upper bounds, percentages versus absolute values, and generic continuous values whose local lexical context is weak. Adding richer deterministic search or compatibility machinery does not by itself close the gap to robust query-level readiness.

The compatibility ablation is also useful for interpreting the robustness patterns from Section 3.3. In the *noisy* variant, the gains are real but much smaller. For TFIDF-TG, *TypeMatch* increases from 0.0520 to 0.1437, while *InstantiationReady* rises only from 0.0211 to 0.0393. This does not indicate that the correction was unimportant; rather, it reflects the structure of the benchmark variant itself. When numeric values are intentionally masked, downstream grounding becomes underdetermined even if the schema is correct. In the *short* variant, the correction again helps, but readiness remains very low because the first sentence often lacks enough contextual and numeric evidence for broad slot filling. The correction therefore resolves a major compatibility problem on the original inputs while also showing that downstream performance still depends strongly on the availability of explicit quantitative evidence.

In summary, even richer deterministic grounding families do not materially surpass the simple TF–IDF typed-greedy baseline. Future improvements should therefore focus less on incremental deterministic search refinements and more on substantially stronger semantic grounding of quantities to parameter roles.

3.5 Statistical Significance and Lexical-Overlap Robustness

Section 3.1 noted that key retrieval and downstream comparisons are supported by paired bootstrap significance tests and by overlap-based stress tests; we report both here.

Using a paired bootstrap with $B = 1{,}000$ resamples over the 331 *orig* test queries, the *InstantiationReady* gap between TF–IDF and Oracle is statistically significant ($\Delta = -0.0393$, 95% CI $[-0.0665, -0.0151]$, $p = 0.004$), confirming that the modest oracle improvement discussed in Section 3.3 is unlikely to be a sampling artifact. The *Schema_R@1* gap between TF–IDF and BM25, recomputed from the corrected

Table 6 Diagnostic error taxonomy for TF–IDF on the *orig* queries. Counts are approximate diagnostic indicators derived from the final measured benchmark and associated reports; categories are not disjoint, and a single query may contribute to multiple categories.

Error type	Count
Wrong schema retrieval	≈ 30
Missing numeric extraction	5
Wrong type assignment (mainly float-related)	230
Wrong slot disambiguation	50
Total vs. unit confusion	20
Min/max inversion	10
Percent vs. absolute confusion	15
Float ambiguity with many candidate values	25
Unsupported schema	0

canonical per-query artifacts, is no longer statistically significant at conventional levels ($\Delta = +0.0211$, 95% CI $[-0.0030, +0.0453]$, $p = 0.088$); this differs from a previously reported marginal significance ($p = 0.022$) that was based on a stale intermediate significance-summary snapshot, and we report the corrected, non-significant result here rather than the earlier one. TF–IDF typed-greedy is statistically indistinguishable from the acceptance-rerank and hierarchical-acceptance-rerank variants ($p = 0.890$ and $p = 0.580$) and from the relation-aware representative ($p = 0.150$), but it significantly outperforms the global-compatibility and ambiguity-aware representatives on *InstantiationReady* ($p < 0.001$ in both tested comparisons; Table 9). This confirms that the negative result for two of the three additional method families – global-compatibility and ambiguity-aware grounding – is not an artifact of sampling noise; the relation-aware family's lower point estimate (0.4985 vs. 0.5287) is directionally consistent with the same pattern but is not itself statistically significant at conventional levels, and we report it as such rather than overstating the evidence.

To test whether strong lexical retrieval mainly reflects benchmark-specific numeric overlap between query and schema text, we re-ran TF–IDF, BM25, and LSA retrieval after stripping numeric tokens and/or stopwords from the query. Table 10 shows that retrieval performance is preserved or improved after sanitization, with LSA showing a pronounced improvement under stopword removal (from 0.8459 to 0.9184, above the canonical TF–IDF baseline). This indicates that retrieval success is driven by structural and domain-term overlap (parameter names, units, problem-type keywords) rather than by matching the specific numeric values in the query. All sanitization experiments were run using the canonical retrieval configuration, including 256-dimensional LSA, so the LSA baseline row reproduces the canonical Table 3 value exactly (0.8459). The TF–IDF and BM25 baseline rows carry a small offset from Table 3 (0.9063 vs. 0.9094 for TF–IDF, 0.8852 vs. 0.8822 for BM25); both sets of values support the same qualitative conclusion, and we disclose the offset rather than

Table 7 Numeric-type compatibility ablation on four representative methods across the three query variants, comparing downstream metrics before and after the numeric-type compatibility correction. The correction mainly affects *TypeMatch* and *InstantiationReady*, while leaving extraction-dependent quantities much less changed.

Var.	Method	$\text{Type}_{\text{before}}$	$\text{Type}_{\text{after}}$	ΔType	$\text{IR}_{\text{before}}$	IR_{after}	ΔIR
orig	TFIDF-TG	0.2497	0.7453	0.4956	0.0695	0.5287	0.4592
orig	TFIDF-ORR	0.2716	0.7016	0.4300	0.0665	0.4411	0.3746
orig	TFIDF-HAR	0.2478	0.7097	0.4619	0.0785	0.5196	0.4411
orig	Oracle-TG	0.2777	0.7998	0.5221	0.0785	0.5680	0.4895
noisy	TFIDF-TG	0.0520	0.1437	0.0917	0.0211	0.0393	0.0182
noisy	TFIDF-ORR	0.0000	0.0000	0.0000	0.0000	0.0000	0.0000
noisy	TFIDF-HAR	0.0576	0.1357	0.0781	0.0242	0.0423	0.0181
noisy	Oracle-TG	0.0620	0.1537	0.0917	0.0211	0.0423	0.0212
short	TFIDF-TG	0.0957	0.2472	0.1515	0.0060	0.0151	0.0091
short	TFIDF-ORR	0.0438	0.0755	0.0317	0.0060	0.0060	0.0000
short	TFIDF-HAR	0.1088	0.2412	0.1324	0.0091	0.0211	0.0120
short	Oracle-TG	0.1143	0.2880	0.1737	0.0030	0.0151	0.0121

Abbreviations: IR = *InstantiationReady*, TFIDF-TG = `tfidf_typed_greedy`, TFIDF-ORR = `tfidf_optimization_role_repair`, TFIDF-HAR = `tfidf_hierarchical_acceptance_rerank`, and Oracle-TG = `oracle_typed_greedy`.

silently reconciling it. This offset is consistent with the 335-document catalog used by this diagnostic script versus the catalog vintage underlying the original Table 3 run (documented in the repository's retrieval-reproducibility notes), and is close in size to the schema-retrieval-rate offset noted later in this section for TFIDF/BM25 in the canonical per-query artifacts, though it does not extend cleanly to LSA. A complementary stratification by query–schema lexical (Jaccard) overlap shows most *orig* queries (327 of 331) fall in a high-overlap bucket with TF–IDF *Schema_R@1* $= 0.9174$; the remaining medium-overlap bucket has only 4 queries (TF–IDF *Schema_R@1* $= 0.0000$ on this bucket) and is too small to support a separate quantitative claim, so we report it only as a qualitative caveat on the generality of the overlap analysis.

Finally, because the central empirical argument of the paper rests on a relatively small gap between TF–IDF and Oracle, we address directly a natural reviewer concern: could an incorrect predicted schema still satisfy the *InstantiationReady* threshold (Eq. (6)) merely because a wrong schema happens to induce a smaller, more easily satisfied eligible-slot set, inflating non-oracle scores relative to what schema-conditioned grounding actually achieves? Recall that *InstantiationReady* does not include exact schema match as an explicit hard gate. To test this, we define a stricter diagnostic variant that adds this gate,

$$\mathit{StrictInstantiationReady}_i = \mathbb{1}[\mathit{SchemaHit}_i \wedge \mathit{Coverage}_i \geq 0.8 \wedge \mathit{TypeMatch}_i \geq 0.8], \tag{7}$$

and report it as a robustness check, not as a replacement for the canonical metric. Table 11 compares both criteria on the four main

Table 8 Representative *orig*-query *InstantiationReady* comparison between the original TF–IDF baseline, the oracle reference, and the strongest representatives of the three additional downstream families. Reported deltas are relative to TFIDF-TG.

Method	InstReady	Δ vs. TFIDF-TG	Note
TFIDF-TG	0.5287	0.0000	baseline
TFIDF-AR	0.5257	-0.0030	near tie
RAL-Basic	0.4985	-0.0302	below baseline
GCG-Full	0.4320	-0.0967	markedly lower
AAG-Beam	0.4230	-0.1057	markedly lower
AAG-Abstain	0.0272	-0.5015	overly conservative
Oracle-TG	0.5680	+0.0393	oracle control

Abbreviations: GCG-Full = `tfidf_global_compat_full`, RAL-Basic = `tfidf_relation_aware_basic`, AAG-Beam = `tfidf_ambiguity_aware_beam`, and AAG-Abstain = `tfidf_ambiguity_aware_abstain`.

Table 9 Paired bootstrap significance tests ($B = 1{,}000$ resamples, *orig* queries) for selected retrieval and downstream comparisons. Diff is method A minus method B on the stated metric.

Comparison (metric)	Diff	95% CI	p
TFIDF vs. BM25 (Schema_R@1)	+0.0211	[-0.0030, +0.0453]	0.088
TFIDF-TG vs. Oracle-TG (InstReady)	-0.0393	[-0.0665, -0.0151]	0.004
TFIDF-TG vs. TFIDF-AR (InstReady)	+0.0030	[-0.0181, +0.0211]	0.890
TFIDF-TG vs. TFIDF-HAR (InstReady)	+0.0091	[-0.0151, +0.0332]	0.580
TFIDF-TG vs. GCG-Full (InstReady)	+0.0967	[+0.0544, +0.1420]	< 0.001
TFIDF-TG vs. RAL-Basic (InstReady)	+0.0302	[-0.0060, +0.0665]	0.150
TFIDF-TG vs. AAG-Beam (InstReady)	+0.1057	[+0.0544, +0.1541]	< 0.001

typed-greedy baselines, computed with the paired-bootstrap methodology (`tools/run_strict_instantiation_ready.py`) from the live per-query artifacts underlying the significance analysis of this section. Its canonical (non-strict) *InstantiationReady* column (0.5287, 0.5196, 0.5076, 0.5680 for TFIDF-TG, BM25-TG, LSA-TG, Oracle-TG) is identical to Table 4, since both are computed directly, and only, from the same canonical per-query files. (During final preparation, Table 4 was found to have been populated from a stale intermediate significance snapshot rather than these live files; we regenerated it from the canonical per-query data, with corrections small in absolute terms – at most 0.009 per metric – that do not change the paper's qualitative conclusions. Part of the TF–IDF/BM25 shift traces to the same schema-retrieval offset discussed above for Table 10, though that explanation does not extend to LSA; see the repository's retrieval-reproducibility notes for the full provenance trace.) Adding the schema-match gate removes 8 of 331 TF–IDF queries (and 9 of 331 BM25, 7 of 331 LSA queries) from the ready count – cases where the predicted

Table 10 Retrieval accuracy (*Schema_R@1*, *orig* queries) after removing numeric tokens and/or stopwords, using the canonical retrieval configuration (256-dimensional LSA). The TF–IDF and BM25 baseline rows carry a small offset from Table 3 (schema-text-construction differences between the two evaluation scripts); the LSA baseline row reproduces Table 3 exactly (see text).

Sanitization	TF–IDF	BM25	LSA
None (baseline)	0.9063	0.8852	0.8459
Numbers removed	0.9063	0.8973	0.8459
Stopwords removed	0.9124	0.9063	0.9184
Numbers & stopwords removed	0.9124	0.9063	0.9184

Table 11 Sensitivity check: canonical *InstantiationReady* versus the stricter *StrictInstantiationReady* (Eq. (7)), *orig* queries, computed from the live per-query artifacts underlying the significance analysis of Section 3.5. The non-strict column here is identical to Table 4, since both are computed from the same canonical per-query artifacts (see text). Δ is standard minus strict; n_{differ} counts queries whose readiness label changes.

Method	InstReady	StrictInstReady	Δ	n_{differ}
TFIDF-TG	0.5287	0.5045	0.0242	8
BM25-TG	0.5196	0.4924	0.0272	9
LSA-TG	0.5076	0.4864	0.0211	7
Oracle-TG	0.5680	0.5680	0.0000	0

schema was wrong but the resulting eligible-slot set still happened to satisfy the coverage and type thresholds – while leaving Oracle unchanged by construction (Oracle always has a correct schema, so the added gate is a no-op for it). The TF–IDF-vs-Oracle gap therefore *widens* under the stricter criterion (from 0.0393 to 0.0634) and remains statistically significant, in fact more so ($p < 0.001$ under the paired bootstrap, versus $p = 0.004$ for the standard metric). This confirms that the modest oracle gain reported throughout this paper is not an artifact of omitting a schema-match gate from *InstantiationReady*; if anything, the schema-conditioned downstream-grounding bottleneck is somewhat understated, not overstated, by the canonical metric.

3.6 Structural and Solver-Backed Validation

The benchmark above is deliberately solver-free: no feasibility test or objective evaluation enters *Schema_R@1*, *Coverage*, *TypeMatch*, or *InstantiationReady*. To assess whether the recovered schema-conditioned structure has any downstream executable value, we report three restricted validation studies on subsets of NLP4LP for which the gold record contains executable solver code (`optimus_code`). These studies use the SciPy HiGHS mixed-integer-linear-programming solver through a compatibility shim;

Table 12 Structural subset (60 instances): schema-hit, structural-validity, and instantiation-completeness rates.

Baseline	Schema Hit	Structural Valid	Inst. Complete
TF–IDF	0.9333	0.7500	0.7500
BM25	0.9000	0.7333	0.7333
Oracle	1.0000	0.7667	0.7833

Gurobi is not required to reproduce any of the results below. All three studies are restricted in scope by construction and are not claims of benchmark-wide executability.

We define the validation metrics once here as per-instance Boolean conditions, each reported as a rate over the relevant subset. *Structural validity* holds when the instantiated schema forms a well-formed linear or mixed-integer program with a defined objective and at least one decision variable, checked statically without invoking a solver. *Instantiation completeness* holds when every scalar slot the gold schema marks as required receives a type-consistent assignment. *Executable* holds when the generated solver program runs to completion without raising an exception. *Solver success* holds when the solver returns a well-defined termination status rather than erroring. *Feasible* holds when the solver reports a feasible solution, and *objective produced* holds when it returns a finite objective value. Structural validity and instantiation completeness are static checks; the remaining four require actually executing the solver.

3.6.1 Structural subset (60 instances).

Table 12 and Figure 2 report schema-hit, structural-validity, and instantiation-completeness rates on a 60-instance subset. TF–IDF reaches a structural-valid rate of 0.75 and an instantiation-complete rate of 0.75; Oracle retrieval reaches 0.7667 and 0.7833, respectively. As in the main benchmark, the oracle gain over TF–IDF is small, reinforcing that downstream grounding – not schema selection – is the binding constraint even on this smaller subset.

3.6.2 Executable-attempt subset (269 instances).

Table 13 reports the largest deterministic subset with non-empty gold `optimus_code` (269 instances). Structural-validity and instantiation-completeness rates remain broadly consistent with the main benchmark (TF–IDF: 0.8141 structural valid, 0.6654 instantiation complete), but the executable, solver-success, feasible, and objective-produced rates are all 0.0 for every baseline. The uniform blocker is a missing `gurobipy` runtime dependency (`ModuleNotFoundError`) in the evaluation environment used for this subset. We report this as a blocker study documenting an environment limitation, not as evidence that the retrieval-assisted grounding method itself produces zero executable value; the final solver-backed subset below shows nonzero outcomes once this dependency is removed by using a SciPy-based shim instead.

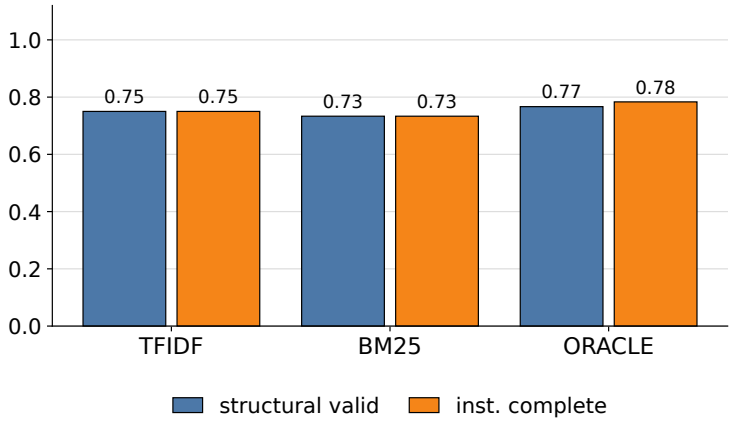


Fig. 2 Structural-validity and instantiation-completeness rates on the 60-instance subset, for TF–IDF, BM25, and Oracle schema retrieval.

Table 13 Executable-attempt subset (269 instances): schema-hit, structural-validity, and instantiation-completeness rates. The executable, solver-success, feasible, and objective-produced rates are uniformly 0.0 for every baseline because of a missing `gurobipy` runtime in this evaluation environment (see text); they are omitted from the table as an environment blocker rather than a scientific result.

Baseline	Schema Hit	Structural Valid	Inst. Complete
TF–IDF	0.9368	0.8141	0.6654
BM25	0.9257	0.8104	0.6580
Oracle	1.0000	0.8253	0.6840

3.6.3 Final solver-backed subset (20 instances).

To obtain a real, non-placeholder solver signal, we further filter to a 20-instance subset selected deterministically by a static shim-compatibility rule (non-empty gold `optimus_code` and predicted code compatible with the SciPy HiGHS shim), capped at 20 instances. Table 14 and Figure 3 report executable, solver-success, feasible, and objective-produced rates for TF–IDF and Oracle retrieval. TF–IDF reaches 0.95 executable, 0.80 solver-success, 0.80 feasible, and 0.80 objective-produced; Oracle reaches 0.95, 0.75, 0.75, and 0.75, respectively. The non-executable rows in each baseline fail due to an indexing incompatibility in the shim path (`IndexError`), affecting 2 of 40 baseline-instance rows across the two baselines. Because this subset is small and selected by compatibility filtering rather than random sampling, these rates should be read as a restricted, pragmatic demonstration that the recovered structure can drive a real solver on compatible cases – not as a benchmark-wide solver-readiness claim.

Table 14 Final solver-backed subset (20 instances, SciPy HiGHS MILP compatibility shim; Gurobi not required).

Baseline	Executable	Solver Success	Feasible	Objective Produced
TF–IDF	0.95	0.80	0.80	0.80
Oracle	0.95	0.75	0.75	0.75

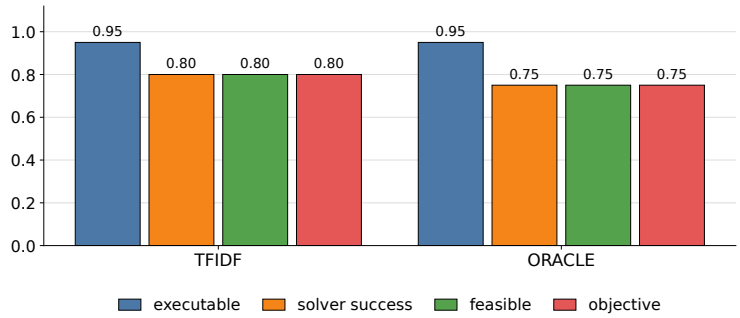


Fig. 3 Executable, solver-success, feasible, and objective-produced rates on the 20-instance final solver-backed subset (SciPy HiGHS shim), for TF–IDF and Oracle schema retrieval.

3.6.4 Summary.

Across all three validation studies, the same qualitative pattern from the main benchmark repeats: Oracle retrieval does not consistently provide a substantial advantage over TF–IDF, and the dominant limitation is downstream. On the structural subset Oracle is modestly higher than TF–IDF (0.7667 vs. 0.75 structural-valid; 0.7833 vs. 0.75 instantiation-complete), but on the final solver-backed subset Oracle is actually *lower* than TF–IDF on solver-success, feasible, and objective-produced rates (0.75 vs. 0.80 on each), a difference plausibly attributable to this subset's small size (20 instances) and its compatibility-filtering rule rather than to a genuine retrieval-quality effect. Taken together, these restricted studies reinforce the main benchmark's conclusion that downstream grounding – not schema selection – is the binding constraint, and they caution against assuming a monotonic Oracle advantage on small subsets. In the executable-attempt subset, missing scalar slots, type mismatch, and incomplete instantiation account for the majority of non-`gurobipy` failures (69, 82, and 267 counted instances, respectively, across baselines), consistent with the error taxonomy in Section 3.4. These restricted studies should be read as evidence of downstream engineering relevance on compatible cases, complementing – not replacing – the main scalar-grounding benchmark.

4 Conclusions and Future Research

This paper studied a retrieval-assisted approach to optimization problem instantiation from natural-language descriptions. Rather than targeting full solver-ready generation,

we focused on a narrower intermediate task: retrieving a compatible optimization schema from a fixed catalog and then instantiating its scalar parameters through deterministic grounding of numeric mentions in the query. This formulation isolates an important intermediate capability in natural-language-to-optimization pipelines: before full model synthesis can be attempted, the system must identify a plausible template and recover at least part of the quantitative information needed to instantiate it.

The main benchmark results lead to a clear conclusion. On the NLP4LP benchmark, schema retrieval is already strong for this task setting: BM25, TF–IDF, and LSA all substantially outperform random selection across the evaluated query variants, with TF–IDF again providing the strongest average top-1 retrieval performance. Thus, within this fixed-catalog benchmark, schema retrieval is not the dominant empirical bottleneck.

The main remaining challenge lies downstream. The benchmark shows that deterministic retrieval-assisted grounding can recover a substantial fraction of usable scalar structure on the original queries, and the strongest non-oracle method, `tfidf_typed_greedy`, achieves *InstantiationReady* = 0.5287 on the 331-query test split. However, the oracle result reaches only 0.5680, so perfect schema selection yields only a modest additional gain over the strongest retrieval-based setting. This indicates that the dominant source of residual error is not selecting the broad problem type, but assigning extracted numeric mentions to the correct scalar roles once a plausible schema has been retrieved. In other words, the main bottleneck in the current pipeline is number-to-slot grounding under semantic ambiguity.

Three richer deterministic families—global compatibility grounding, relation-aware linking, and ambiguity-aware grounding—none surpassed the `tfidf_typed_greedy` baseline on the original benchmark inputs. At the same time, the numeric-type compatibility ablation shows that correcting how float-valued slots are matched produces large gains in *TypeMatch* and *InstantiationReady* while leaving extraction-dependent quantities much less changed. Together, these findings indicate that the remaining difficulty is not simply a lack of downstream engineering variety, but a more fundamental challenge of semantically determining which quantity in the query plays which optimization role.

Beyond the core benchmark, the structural and solver-backed validation studies in Section 3.6 give a more practical perspective. TF–IDF and Oracle retrieval produce structurally valid, instantiation-complete artifacts on 75–78% of the 60-instance structural subset (Table 12); the 269-instance executable-attempt subset documents a transparent environment blocker (a missing `gurobipy` runtime) rather than a property of the method (Table 13); and on the 20-instance final solver-backed subset, TF–IDF reaches 0.80 solver-success, feasible, and objective-produced rates versus 0.75 for Oracle (Table 14) – Oracle is not consistently better across the three studies, again underscoring that downstream grounding, not schema selection, is the binding constraint. These subsets are intentionally narrow and should not be read as benchmark-wide solver readiness, but they show the recovered structure can drive a real solver on compatible cases.

The contribution of the paper should therefore be understood in a measured way: not a complete natural-language-to-optimization compiler, nor a replacement for an optimization modeler, but a transparent retrieval-and-instantiation component with measurable downstream value for intelligent decision support, in the sense already discussed in Section 1.4.

4.1 Limitations and Threats to Validity

The scope restrictions adopted throughout this paper (Section 2.1) are deliberate design choices, but they also bound what the results can and cannot support. We summarize the main limitations and threats to validity here rather than leaving them dispersed across the paper.

Fixed-catalog retrieval. Schema retrieval is evaluated only as a 335-way identification problem over the benchmark-induced catalog described in Section 2.2. The strong retrieval numbers in Section 3.2 characterize this fixed-catalog setting and should not be read as evidence that lexical retrieval would remain competitive against an open-domain or substantially larger schema library, where semantic rather than lexical retrieval may become necessary.

Dataset dependence and lexical overlap. All quantitative results are computed on NLP4LP, a single benchmark. The overlap-based stress tests in Section 3.5 show that retrieval accuracy is not simply an artifact of numeric-value overlap between query and schema text, but the diagnostic medium-overlap bucket contains only 4 of 331 *orig* queries – too few to fully rule out benchmark-specific lexical or structural regularities at scale. Generalization to optimization problem descriptions written in different styles, domains, or languages is untested.

Gated data access. NLP4LP requires an approved Hugging Face account and access token (Declarations, Data availability). Independent researchers can inspect and rerun analyses on the committed result artifacts without this access, but a full end-to-end rerun on live benchmark queries requires it, which is a real (if benchmark-typical) reproducibility constraint.

Scalar-only instantiation. The benchmarked instantiation stage recovers only scalar parameters; list-valued, vector-valued, and dictionary-valued parameters are out of scope (Section 2.4). The reported *Coverage*, *TypeMatch*, and *Instantiation-Ready* scores therefore characterize a restricted parameter-recovery problem, not full model-parameter reconstruction.

Heuristic type and role inference. Both coarse type inference (Section 2.3) and optimization-role compatibility (Section 2.3) are deterministic, lexicon- and rule-based mechanisms with no formal correctness guarantee. The error taxonomy (Table 6) and the negative result for three additional richer deterministic families (Section 3.4) indicate that this heuristic layer is also the paper's main open problem: semantic quantity-to-role ambiguity is not resolved by the methods evaluated here.

Restricted and small solver-backed validation. The structural and solver-backed studies in Section 3.6 use restricted subsets (60, 269, and 20 instances, respectively) selected by data-availability or compatibility filtering rather than random sampling, and the only solver evaluated is SciPy's open-source HiGHS backend; Gurobi and

other commercial or specialized solvers are not exercised. The 20-instance final solver-backed subset in particular is too small to support benchmark-wide solver-readiness claims, and the 269-instance executable-attempt subset reports a uniform environment blocker (a missing `gurobipy` runtime) rather than a fully executed solver comparison.

No direct experimental comparison against end-to-end LLM systems. Section 1.2 discusses several recent end-to-end large-language-model-based optimization-modeling systems (e.g., OptiMUS, OptLLM, Chain-of-Experts, LLMOPT, OPT2CODE) qualitatively, but this paper does not run them head-to-head against the proposed pipeline on NLP4LP. These systems target a different, broader task (full formulation and solver-code generation) and typically depend on paid API access or substantial compute, which was outside the scope and budget of this study; a controlled quantitative comparison is left to future work.

No external dataset validation or deployment study. Although Section 1.2 discusses Text2Zinc and CP-Bench as related benchmarks, the framework is not evaluated on them or on any other external dataset in this paper, so cross-benchmark generalization is untested. Similarly, no user study or production deployment was conducted; the practical-usefulness claims in this paper are limited to the restricted, benchmark-scoped, and subset-based evidence reported in Sections 3.3–3.6.

4.2 Future Research Directions

Several directions for future research follow naturally from these findings. The first is to strengthen the downstream grounding stage itself with substantially stronger semantic role-sensitive methods, rather than additional variants of the current deterministic assignment family. The second is to extend the current scalar-only setting to more structured parameter types, including vectors, indexed collections, and dictionary-valued arguments. The third is to explore hybrid systems in which transparent schema retrieval provides structural scaffolding for stronger learned or large-language-model-based downstream grounding modules. A fourth direction is to expand the current restricted engineering validations into broader executable studies with more complete solver-backed coverage and wider template support.

More broadly, the results suggest that intermediate capabilities such as schema retrieval, schema-conditioned slot selection, and partial structured instantiation deserve independent attention in the design of transparent optimization support systems. Full natural-language-to-optimization automation remains difficult, but intermediate systems can already provide useful support by narrowing candidate problem structure and recovering part of the quantitative content needed for formal modeling. At the same time, the present results make clear that robust quantity-to-role grounding remains unresolved even when retrieval is strong. This makes the intermediate task studied here both practically relevant and scientifically valuable as a benchmarked subproblem in natural-language-to-optimization research.

Statements and Declarations

Funding. This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Competing interests. The author declares that there are no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.
Ethics approval and consent to participate. Not applicable; this study did not involve human participants, human data, or animals.
Data availability. The benchmark used in this paper, NLP4LP (`udell-lab/NLP4LP` on Hugging Face), is a gated third-party dataset that requires an approved Hugging Face account and access token; it is not redistributed by the author. The committed camera-ready tables, figures, and analysis reports summarizing results computed from this dataset are available at https://github.com/SoroushVahidi/combinatorial-opt-agent (see `results/` and `analysis/`).
Code availability. The retrieval, grounding, evaluation, and figure/table-generation code used in this paper is available at https://github.com/SoroushVahidi/combinatorial-opt-agent. Reading the committed tables and figures, and running the test suite, do not require Hugging Face dataset access; reproducing a full end-to-end rerun on NLP4LP (retrieval and grounding on the live benchmark) requires the gated dataset access described above.
Author contributions. Soroush Vahidi: Conceptualization, Methodology, Software, Validation, Formal analysis, Investigation, Data curation, Writing – original draft, Writing – review & editing, Visualization.

References

[1] AhmadiTeshnizi A, Gao W, Udell M (2023) NLP4LP: A dataset for natural language to linear programming and mixed-integer linear programming. Hugging Face dataset repository, URL https://huggingface.co/datasets/udell-lab/NLP4LP, available at the udell-lab/NLP4LP dataset page

[2] AhmadiTeshnizi A, Gao W, Brunborg H, et al (2024) OptiMUS-0.3: Using large language models to model and solve optimization problems at scale. arXiv preprint arXiv:240719633 https://doi.org/10.48550/arXiv.2407.19633, URL https://arxiv.org/abs/2407.19633

[3] AhmadiTeshnizi A, Gao W, Udell M (2024) OptiMUS: Scalable optimization modeling with (MI)LP solvers and large language models. In: Proceedings of the 41st International Conference on Machine Learning, Proceedings of Machine Learning Research, vol 235. PMLR, pp 577–596, URL https://proceedings.mlr.press/v235/ahmaditeshnizi24a.html

[4] Ahmed T, Choudhury S (2025) OPT2CODE: A retrieval-augmented framework for solving linear programming problems. Natural Language Processing Journal 13:100185. https://doi.org/10.1016/j.nlp.2025.100185, URL https://www.sciencedirect.com/science/article/pii/S2949719125000615

[5] Biemans B, Troubil P, Grau I, et al (2026) Explainable optimization: Leveraging large language models for user-friendly explanations. In: Guidotti R, Schmid U,

Longo L (eds) Explainable Artificial Intelligence, Communications in Computer and Information Science, vol 2578. Springer, Cham, p 44–67, https://doi.org/10.1007/978-3-032-08327-2_3, URL https://link.springer.com/chapter/10.1007/978-3-032-08327-2_3, xAI 2025, first online 12 October 2025

[6] Deerwester S, Dumais ST, Furnas GW, et al (1990) Indexing by latent semantic analysis. Journal of the American Society for Information Science 41(6):391–407. https://doi.org/10.1002/(SICI)1097-4571(199009)41:6⟨391::AID-ASI1⟩3.0.CO;2-9

[7] Duchnowski A, Pavlick E, Koller A (2025) A knapsack by any other name: Presentation impacts LLM performance on NP-hard problems. In: Findings of the Association for Computational Linguistics: EMNLP 2025. Association for Computational Linguistics, Suzhou, China, pp 6628–6651, https://doi.org/10.18653/v1/2025.findings-emnlp.352, URL https://aclanthology.org/2025.findings-emnlp.352/

[8] Huang S, Yang K, Qi S, et al (2024) When large language model meets optimization. Swarm and Evolutionary Computation 90:101663. https://doi.org/10.1016/j.swevo.2024.101663, URL https://www.sciencedirect.com/science/article/pii/S2210650224002013

[9] Huang X, Shen Q, Hu Y, et al (2025) LLMs for mathematical modeling: Towards bridging the gap between natural and mathematical languages. In: Chiruzzo L, Ritter A, Wang L (eds) Findings of the Association for Computational Linguistics: NAACL 2025. Association for Computational Linguistics, Albuquerque, New Mexico, pp 2678–2710, https://doi.org/10.18653/v1/2025.findings-naacl.146, URL https://aclanthology.org/2025.findings-naacl.146/

[10] Jiang C, Shu X, Qian H, et al (2025) LLMOPT: Learning to define and solve general optimization problems from scratch. In: Proceedings of the Thirteenth International Conference on Learning Representations, URL https://openreview.net/forum?id=9OMvtboTJg

[11] Kadioglu S, Dakle PP, Uppuluri K, et al (2024) Ner4Opt: Named entity recognition for optimization modelling from natural language. Constraints 29(3–4):261–299. https://doi.org/10.1007/s10601-024-09376-5

[12] Lhoest Q, Villanova del Moral A, Jernite Y, et al (2021) Datasets: A community library for natural language processing. In: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pp 175–184, https://doi.org/10.18653/v1/2021.emnlp-demo.21, URL https://aclanthology.org/2021.emnlp-demo.21/

[13] Michailidis K, Tsouros D, Guns T (2025) CP-Bench: Evaluating large language models for constraint modelling. arXiv preprint arXiv:250606052 https://doi.org/10.48550/arXiv.2506.06052, URL https://arxiv.org/abs/2506.06052

[14] Pedregosa F, Varoquaux G, Gramfort A, et al (2011) Scikit-learn: Machine learning in Python. Journal of Machine Learning Research 12:2825–2830

[15] Powell C, Riccardi A (2025) Generating textual explanations for scheduling systems leveraging the reasoning capabilities of large language models. Journal of Intelligent Information Systems 63(4):1287–1337. https://doi.org/10.1007/s10844-025-00940-w, URL https://doi.org/10.1007/s10844-025-00940-w

[16] Ramamonjison R, Li H, Yu TTL, et al (2022) Augmenting operations research with auto-formulation of optimization models from problem descriptions. In: Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track. Association for Computational Linguistics, pp 29–62, https://doi.org/10.18653/v1/2022.emnlp-industry.4, URL https://aclanthology.org/2022.emnlp-industry.4/

[17] Ramamonjison R, Yu T, Li R, et al (2023) NL4Opt competition: Formulating optimization problems based on their natural language descriptions. In: Proceedings of the NeurIPS 2022 Competitions Track, Proceedings of Machine Learning Research, vol 220. PMLR, pp 189–203, URL https://proceedings.mlr.press/v220/ramamonjison23a.html

[18] Ramamonjison R, Yu T, Xing L, et al (2023) LaTeX2Solver: A hierarchical semantic parsing of LaTeX document into code for an assistive optimization modeling application. In: Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations). Association for Computational Linguistics, pp 471–478, https://doi.org/10.18653/v1/2023.acl-demo.45, URL https://aclanthology.org/2023.acl-demo.45/

[19] Robertson S, Zaragoza H (2009) The probabilistic relevance framework: BM25 and beyond. Foundations and Trends in Information Retrieval 3(4):333–389. https://doi.org/10.1561/1500000019

[20] Singirikonda A, Kadioglu S, Uppuluri K (2025) Text2Zinc: A cross-domain dataset for modeling optimization and satisfaction problems in MiniZinc. arXiv preprint arXiv:250310642 https://doi.org/10.48550/arXiv.2503.10642, URL https://arxiv.org/abs/2503.10642

[21] Xiao Z, Zhang D, Wu Y, et al (2024) Chain-of-experts: When LLMs meet complex operations research problems. In: Proceedings of the Twelfth International Conference on Learning Representations, URL https://openreview.net/forum?id=HobyL1B9CZ

[22] Xiao Z, Xie J, Xu L, et al (2025) A survey of optimization modeling meets LLMs: Progress and future directions. In: Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, pp 10742–10750, URL https://www.ijcai.org/proceedings/2025/1192

[23] Zhang J, Wang W, Guo S, et al (2024) Solving general natural-language-description optimization problems with large language models. In: Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 6: Industry Track). Association for Computational Linguistics, pp 483–490, https://doi.org/10.18653/v1/2024.naacl-industry.42, URL https://aclanthology.org/2024.naacl-industry.42/