Interactive Optimization Modeling via Preference Intermediate Representation: Automating Domain Adaptation from Raw Data

1st Li Chen
Department of Computer Science and Technology
Tsinghua University
Beijing, China
lizchan0426@gmail.com

2nd Chun Yu
Department of Computer Science and Technology,
BNRist, College of AI
Tsinghua University
Beijing, China
chunyu@tsinghua.edu.cn

***Abstract*—Existing natural language-to-optimization-model approaches face significant challenges in real-world scenarios, including handling large-scale heterogeneous data and accommodating evolving user preferences. To address these, we propose the Preference Intermediate Representation (PIR), which decouples problem data from preference logic and separates preference models from implementation code. PIR supports the iterative refinement of user preferences within the intermediate layer and enhances the scalability and flexibility of optimization modeling. Furthermore, we introduce an automated framework to generate domain-specific PIR components (Item Selectors and Preference Expressions) from raw problem data and task descriptions. We evaluate our approach in the domain of student course timetabling. Experimental results demonstrate that our framework can effectively generate high-quality domain-specific PIR components. Furthermore, when integrated into an interactive dialogue system, the PIR-based natural language parsing approach achieves 93.50% exact accuracy, significantly outperforming baselines.**

***Index Terms*—Interactive Optimization Modeling, Large Language Models, Preference Intermediate Representation, Natural Language Interfaces**

I. Introduction

Complex decision and planning tasks are ubiquitous in real-world scenarios, ranging from course timetabling [1] to travel planning [2], [3]. These tasks are characterized by high-dimensional search spaces, multiple constraints, and the need to balance competing objectives. For instance, in student course timetabling, students must select a subset of courses from a large pool offered by the university, considering factors such as curriculum requirements, preferred instructors, and the distribution of course times to create an optimal timetable. Similarly, in travel planning, travelers aim to select the best destinations, routes, and activities based on their budget, time constraints, and personal preferences, while also factoring in variables like weather, availability, and travel duration. The combinatorial nature of these problems makes manual optimization not only tedious but also prone to sub-optimal outcomes.

A common approach is to model these tasks as optimization problems, which can then be solved using mathematical solvers [4]. Traditionally, this process relies on domain experts to conduct extensive requirement gathering and develop decision support systems with predefined templates [1]. Users are typically limited to adjusting numerical parameters within these rigid systems. However, this approach suffers from a significant knowledge engineering bottleneck: the heavy reliance on manual labor for system development makes it difficult to scale across different domains or adapt to the unique needs of individual users.

The recent advancement of Large Language Models (LLMs) has opened new avenues for automatic optimization modeling [5]–[7] , where LLMs are used to generate mathematical models directly from natural language descriptions. However, a significant assumption underlying these approaches is that the natural language description contains a complete and precise formulation of the problem. In practice, this assumption often does not hold true. Specifically, real-world scenarios present several challenges:

1) *Large Problem Data:* Real-world tasks involve large-scale candidate sets with heterogeneous attributes. Explicitly describing these vast problem data within a natural language prompt is hindered by context window limitations, which leads to significant information loss.
2) *Constructive Preferences:* User preferences are often not fully defined at the outset. Instead, they evolve over time based on the results during the decision-making or planning process.

To address these challenges, we propose the **Preference Intermediate Representation (PIR)**, a modular framework designed to: (1) decouple the problem data from preference logic, mitigating the issue of large-scale data heterogeneity, and (2) decouple the preference model from implementation code, supporting the iterative nature of constructive preferences. This approach overcomes the limitations of direct natural language-to-model generation by introducing a structured intermediate representation.

To enable the rapid customization of this interactive system for new domains, we further propose an automated frame-

work for generating the domain-specific Item Selectors and Preference Expressions based on raw problem data and task descriptions. It employs a verification-driven generation mechanism, utilizing hybrid sampling and dual-code verification (comparing solver code against a semantic oracle) to ensure the correctness and robustness of the generated components.

We instantiate and evaluate our approach in the domain of university course timetabling. Experimental results show that our automated framework can effectively generate domain-specific item selectors and preference expressions. Additionally, we implement a dialogue system to collect real user preference instructions and map them to the three components of PIR, demonstrating that PIR can effectively represent and handle complex user instructions. Furthermore, we conduct a comparative experiment on the natural language parsing of the collected user instructions. The results show that our PIR-based approach achieves a 93.50% exact accuracy, significantly outperforming existing baselines. Besides, we additionally conduct a case study on travel planning to explore the framework's applicability to a domain with heterogeneous multi-type candidate structures.

In summary, our contributions are as follows:

- We introduce the Preference Intermediate Representation (PIR), a structured formalism that decouples problem data, preference logic, and optimization conditions, facilitating interactive and reusable optimization modeling.
- We design an automated framework that synthesizes domain-specific PIR components from raw problem data and task descriptions.
- We demonstrate the effectiveness of our approach through comprehensive evaluation on student course timetabling.

Our code is publicly available at https://github.com/Anony-hci/OptiLens.git.

II. Related Work: LLMs and Optimization

Initially, Large Language Models (LLMs) have been tested for their direct planning capabilities in travel planning systems [3]. However, experimental results indicate that current LLMs still struggle with complex planning scenarios. Even GPT-4 achieves only around a 0.6% success rate in such tasks. Subsequently, the combination of LLMs with optimization solvers [2], [4], [8] has emerged as a more widely adopted approach to support robust end-user planning. In this framework, LLMs act as semantic parsers, translating natural language inputs into machine-understandable representations and explaining the solver's output. Optimization solvers (e.g., SAT solvers, Gurobi) then handle the computational tasks. As a result, the core of planning tasks has shifted to optimization modeling, where both the task and user preferences are modeled as optimization problems.

Existing work has proposed several benchmarks for converting natural language descriptions into optimization problems. The NL4Opt competition [5] introduced benchmark tasks for semantic parsing and logical form generation from optimization problem descriptions. Building on this, ComplexOR [6] and IndustryOR [7] extended the framework to

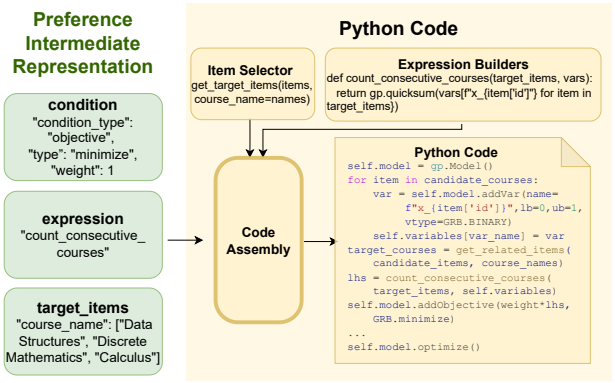

Fig. 1. The workflow of generating executable optimization code via Preference Intermediate Representation (PIR). A structured PIR instance (left) acts as a blueprint, directing the Code Assembly module to invoke the corresponding Item Selector and Expression Builder (top right). This process dynamically assembles the final solver code (bottom right), effectively decoupling the preference model from the underlying implementation details.

industrial scenarios involving implicit constraints and domain-specific knowledge. OptiMUS-0.3 [9] tackled challenges of long, large-scale textual descriptions by a modular framework to separate data representation from natural language input. Additionally, MaMo [10] provided a broad evaluation of LLMs' abilities to generate mathematical models across various optimization problem types. These efforts establish a foundation for reliably converting natural language problem descriptions into formal optimization models.

Building on these modeling capabilities, several interactive systems have adopted the LLM and optimization framework to support practical decision-making. Systems like TTG [2] and LLM-SMT [8] demonstrate this workflow for travel planning, while MeetMate [4] applies it to collaborative meeting scheduling. Other systems, such as OptiChat [11], [12] and OptiGuide [13], focus on helping users interpret solver outputs, diagnose infeasibility, and explore what-if scenarios.

III. Preference Intermediate Representation (PIR)

We design the Preference Intermediate Representation (PIR) to achieve two key decouplings:

1) Decoupling the problem data from the preference expression: To address this, we introduce an item selector that filters the target decision variables from the problem data for preference modeling. This approach has two key advantages:
   - It effectively adapts to changes in the user's candidate options during the preference modeling process, without requiring modifications to the preference expression itself.
   - It reduces the complexity of representing preference concepts by decomposing the monolithic preference into two independent modules: an item selector that handles variable filtering and an expression function that captures mathematical logic. This modular

decomposition lowers the complexity of code generation and enhances the reusability of preference components.

2) Decoupling the preference model from the implementation code: This decoupling enables users to interactively modify the preference representation within the PIR, without needing to alter the underlying implementation code. When the optimization problem needs to be solved, the PIR list is used to generate the corresponding optimization model code, which can then be used for problem-solving.

Specifically, each preference π is defined as a triplet:

$$\pi = \langle \phi_{cond}, \phi_{item}, \phi_{expr} \rangle \tag{1}$$

*1) **Optimization Condition** (ϕ_{cond}):* The optimization condition consists of three types, with each optimization condition type τ_{type} including different parameters θ_{val}.

1) **Objective (τ_{type} = obj):** Treats $\mathcal{E}$ as a term to be optimized. The parameter is defined as a tuple $\theta_{val} = \langle d, w \rangle$:
   - $d \in \{\text{MIN}, \text{MAX}\}$ specifies the optimization direction.
   - $w \in \mathbb{R}^+$ represents the weight coefficient, allowing for multi-objective scalarization (i.e., adding $w \cdot \mathcal{E}$ to the global objective function).
2) **Constraint (τ_{type} = con):** Treats $\mathcal{E}$ as a hard restriction on the feasible region. The parameter is defined as a tuple $\theta_{val} = \langle \bowtie, b \rangle$:
   - $\bowtie \in \{\leq, =, \geq\}$ denotes the relational operator.
   - $b \in \mathbb{R}$ represents the threshold value (RHS).

   The solver enforces the logical proposition: $\mathcal{E} \bowtie b$.
3) **Item Priority (τ_{type} = prio):** The parameter $\theta_{val} = p \in \mathbb{R}$ represents the priority assigned to the target candidate items $\mathbf{x}_{target}$.

   Mathematically, this injects a linear term into the global objective function for each variable $x_k \in \mathbf{x}_{target}$:

$$\mathcal{J}_{global} \leftarrow \mathcal{J}_{global} + \sum_{x_k \in \mathbf{x}_{target}} (p \cdot x_k) \tag{2}$$

   This mechanism allows users to express granular relative preferences (e.g., "prioritizing Instructor A over Instructor B") by assigning differential priority scores to their respective data entities.

*2) **Item Selector** (ϕ_{item}):* The item selector grounds preference into specific decision variables by filtering based on domain attributes. Let $\mathcal{D}$ be the *Problem Data*, where each entry $d_i \in \mathcal{D}$ contains a set of attributes (e.g., `instructor`, `time`) corresponding to a decision variable $x_i \in \mathcal{X}$.

Formalized, a selector ϕ_{item} encapsulates a specific *filtering predicate* ψ. It retrieves the subset of variables whose associated data attributes satisfy this condition:

$$\mathbf{x}_{target} \leftarrow \{x_i \in \mathcal{X} \mid \psi(d_i) \text{ is True}\} \tag{3}$$

For example, to retrieve "classes assigned to Professor A on Mondays," the predicate would be defined as:

$$\psi(d_i) := (d_i.\texttt{instructor} == \text{"Prof. A"}) \wedge (d_i.\texttt{day} == \text{"Mon"})$$

This mechanism ensures that variable selection is dynamically driven by the underlying data schema rather than hard-coded indices.

*3) **Expression Function** (ϕ_{expr}):* The expression function encapsulates the mathematical logic of the preference concept. It acts as a mapping function that transforms the target variables into a linear symbolic expression $\mathcal{E}$:

$$\mathcal{E} \leftarrow \phi_{expr}(\mathbf{x}_{target}, \theta) \tag{4}$$

where θ represents optional hyperparameters (e.g., weights). Crucially, ϕ_{expr} is domain-agnostic regarding the constraint type; it simply computes a value (e.g., "count 8 AM courses" or "sum of credits").

A. Problem Model

The *Problem Model* $\mathcal{M}$ is formally defined as a set of PIR instances:

$$\mathcal{M} = \{\pi_1, \pi_2, \ldots, \pi_N\} \tag{5}$$

This structure enables granular interactivity. When a user refines a requirement (e.g., changing a constraint from "less than 5" to a "minimize" objective), the system only needs to update the τ_{cond} component of the corresponding π_i, while leaving the *Item Selector* and *Expression Function* unchanged.

B. Solver Code Generation

Our framework is compatible with various optimization solvers, generating the corresponding executable code based on $\mathcal{M}$. The code generation process follows a standard protocol:

1) **Context Initialization:** Initialize the solver environment and global variables $\mathcal{X}$.
2) **Constraint/Objective Injection:** For each $\pi_i \in \mathcal{M}$:
   a) Execute ϕ_{item} to bind variables.
   b) Execute ϕ_{expr} to construct the linear expression $\mathcal{E}_i$.
   c) Apply τ_{cond} to add the corresponding constraint or objective term to the solver model.
3) **Solving:** Trigger the optimization process.

IV. Framework for Generating Domain-Specific Item Selectors and Preference Expressions

To implement an interactive optimization modeling system for a specific domain under the PIR design, the system must define domain-specific item selectors and preference expressions. To automate this, we propose a framework that autonomously generates these components from problem data and task descriptions. As illustrated in Figure 2, our framework consists of several phases:

A. Phase 1: Domain Analysis and Context Establishment

Before synthesizing executable components, the framework conducts a semantic analysis of the problem domain to define the necessary context and constraints.

1) **Global Constraint Analyst:** This agent infers *global implicit constraints* from the task description, ensuring that the generated model results in feasible solutions.

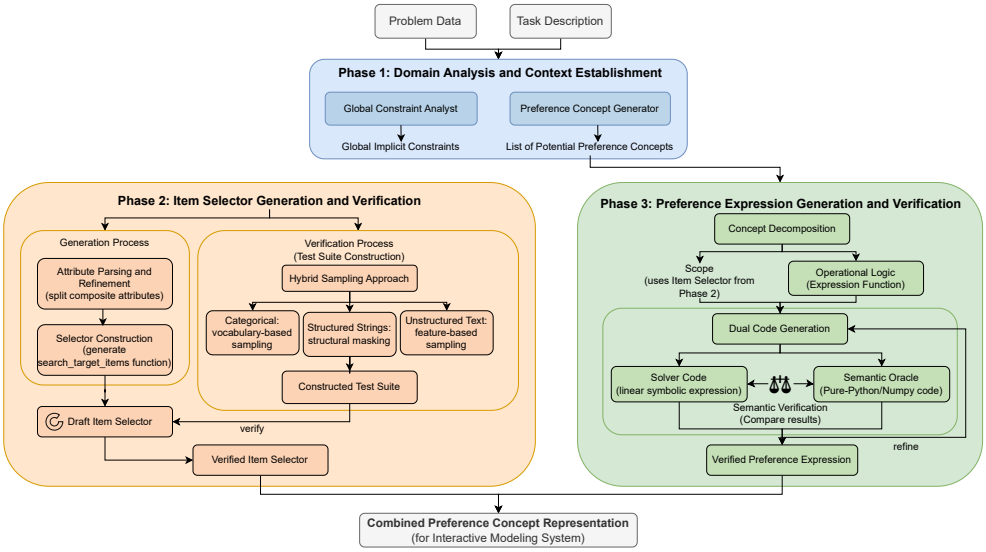

Fig. 2. Framework for automated domain-specific customization. The framework synthesizes interactive optimization modeling components from raw problem data and natural language task descriptions through three sequential phases. Phase 1 establishes domain context by analyzing global constraints and generating potential preference concepts. Phase 2 focuses on the generation and rigorous validation of Item Selectors, employing a hybrid sampling approach to construct diverse test suites. Phase 3 handles Preference Expression generation through concept decomposition and a dual-code generation mechanism. Semantic correctness is ensured by verifying the linear symbolic solver code (f_{milp}) against a parallel Python-based semantic oracle (f_{oracle}) using synthesized data. The verified selectors and expressions are finally integrated into a combined representation for interactive optimization modeling systems.

These constraints typically represent domain-specific limitations, such as non-overlapping time slots in student course timetabling tasks.

2) **Preference Concept Generator:** Simultaneously, this agent generates a list of potential preference concepts (e.g., "early 8 AM classes," "the number of days with classes"). This list serves as the foundation for generating preference expressions in Phase 3.

B. Phase 2: Item Selector Generation and Validation

The goal of the Item Selector is to provide granular control over the problem data via its attributes. Each data attribute should have a corresponding filter to facilitate selection. However, raw problem data may include composite attributes that need to be split into their individual components. For example, the attribute `Time` with the value "3-2(Full)" contains multiple elements, such as the weekday, period, and the specific weeks, which need to be parsed and treated separately.

The prompt design of the *Item Selector* generation is as follows:

1) **Data Description:** For each attribute, we first select sample data with significant variations to summarize the key characteristics of the data.
2) **Attribute Parsing and Refinement:** The LLM constructs appropriate filters for each attribute, including matching operators and specific values. Additionally, for composite fields, the system splits them to extract fine-grained attributes.
3) **Selector Construction:** The system generates the `search_target_items` function, where the extracted attributes are exposed as independent filter parameters. This allows the system to translate high-level preferences (e.g., "Tuesday mornings") into precise filter criteria, such as {weekday: 2, periods: [1,2]}, which are then applied to the decomposed data structure.

To validate the generated *Item Selector*, we construct a test suite from the domain dataset using a hybrid sampling approach:

- **For Categorical Attributes:** Fields like `Department` contain a finite set of fixed values. Here, we employ *vocabulary-based sampling*.
  1) *Extraction:* We extract the set of unique values (the vocabulary) present in the column (e.g., {"Mathematics", "Physics", "CS", `NaN`}).
  2) *Coverage:* We sample one record for each unique category. This ensures the selector is verified against every valid option in the domain, preventing errors where specific categories (e.g., containing spaces or special characters) might be mishandled.
- **For Structured or Semi-Structured Strings:** Fields like `Time` often exhibit complex but finite patterns. We apply *structural masking*:
  1) *Masking:* We map raw values to abstract format signatures (e.g., converting digits to `d` and letters to `w` while preserving delimiters like ";").
  2) *Stratification:* We group records by their unique signature and sample one representative per group. This ensures coverage of all syntactic variations (e.g., single-slot vs. multi-slot schedules).
- **For Unstructured Text:** For free-text fields, we employ *feature-Based sampling*, selecting records with extreme

features (e.g., empty strings, maximum length, special characters) to stress-test string matching operations.

C. Phase 3: Preference Expression Generation and Verification

For the preference concepts generated in Phase 1, we first decompose them into two components: the scope (Item Selector) and the operational logic (Expression Function).

- For example, from the concept "*8 AM classes*," the system isolates the scope {`Period=1`} and extracts the aggregation logic `count_courses` as the expression.
- From the concept *Avoid scheduling Mathematics and Physics consecutively*," the scope is {`Department=Math,Department=Physics`}, and the expression is `count_consecutive_courses`.

For any extracted Expression Functions, if they do not already exist in the current library, the system will generate them. Since the solver code may involve linearizing the expression, which could introduce semantic errors, we also generate reference code to facilitate rigorous self-verification.

- **The Solver Code (f_{milp}):** The primary implementation is compatible with the optimization solver (e.g., Gurobi) and maps decision variables to a linear symbolic expression. This process may involve complex formulation techniques, such as introducing auxiliary variables to linearize non-linear objectives, ensuring compatibility with the solver's syntax.
- **The Semantic Oracle (f_{oracle}):** A parallel pure-Python implementation that uses standard numerical libraries (e.g., NumPy) to operate on concrete data values. Crucially, it serves as the Ground Truth to verify the semantic correctness of f_{milp}. By comparing the results of both functions on synthesized data, the system ensures that the solver code aligns with the intended logic, going beyond simple syntax or runtime error checking.

These Item Selectors and Preference Expressions are combined to represent the preference concept, enabling interactive modeling tasks.

V. Evaluation: A Case Study on Student Course Timetabling

To evaluate the proposed multi-agent framework and assess the effectiveness of the Preference Intermediate Representation (PIR) design, we conduct an in-depth case study on a complex, real-world domain: Student Course Timetabling. We use real course data from our university for the Spring 2025 semester. In this scenario, the multi-agent framework generates the PIR component library, and we develop an interactive dialogue system that supports students in adding courses of interest, expressing their preferences through dialogue, and displaying feasible timetable solutions. Our system is implemented by calling the OpenAI GPT-4o-08-16 API.

Our evaluation is guided by three overarching research questions:

- **RQ1 (Generative Capability):** Can the multi-agent framework autonomously leverage raw problem data and task descriptions from the complex course timetabling domain to construct a high-quality library of domain-specific Item Selectors and Preference Expressions?
- **RQ2 (Representational Effectiveness):** To what extent can the proposed PIR design effectively characterize the diverse preference descriptions expressed by users in natural language?
- **RQ3 (Comparative Performance):** How does the PIR-based natural language parsing approach compare to end-to-end baselines in terms of parsing accuracy?

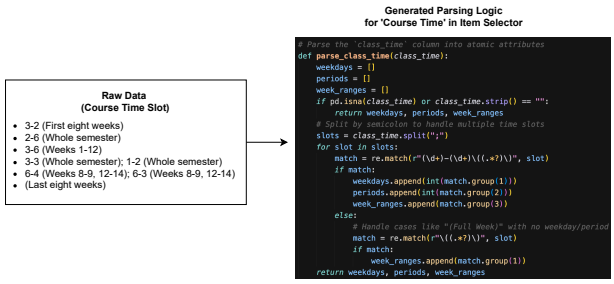

Fig. 3. Example of Item Selector generation for the "Course Time" attribute. The raw data consists of various time slots, and the system parses these into atomic attributes, such as weekdays, periods, and week ranges, using a generated parsing logic.

A. RQ1: Effective Generation of Item Selectors and Preference Expressions

We ran the multi-agent system three times to generate the corresponding Item Selectors and Preference Expressions, and each time, the Item Selector passed the sampled test suite (n=300). These results demonstrate that our framework can effectively generate the Item Selector. The course data includes six columns: course name, instructor, time, credits, department, and dropout rate. For Item Selectors, the system can efficiently parse simple string attributes (e.g., course name, instructor) and numerical values (e.g., dropout rate) using a single-pass generation approach. For more complex or heterogeneous attributes, such as course time, the system, after being provided with sufficiently varied sample data, is able to generate effective parsing functions. As shown in Figure 3, the system successfully handles the decomposition of course time data into granular components like weekdays, periods, and week ranges.

The preference expressions generated by the system are limited, primarily because the LLMs brainstorm potential user preferences that are mainly at the attribute level. These preferences mostly relate to specific courses or course lists, which can be modeled by accumulating courses or assigning priorities to particular courses. For example:

1) "Try to avoid classes on Friday; I want to go home early."
2) "I have trouble waking up in the morning, so it would be better if I had classes in the afternoon or evening."
3) "I need 2 more credits to graduate; please find suitable courses."
4) "Try to avoid classes with Professor Wang, I heard he gives low grades."

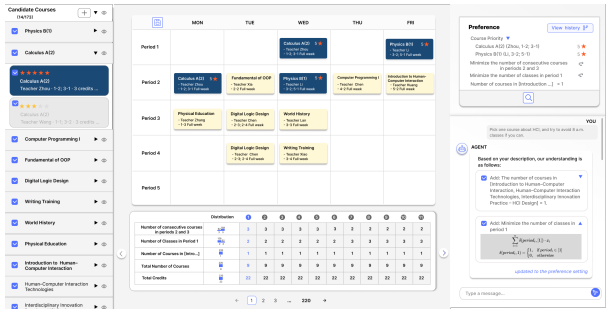

Fig. 4. The user interface of the dialogue system for student course timetabling, enabling users to add candidate courses, express preferences through dialogue, and display feasible timetable solutions.

5) "Are there any courses related to aesthetics? I'm really interested in that."
6) "Give priority to courses with a low dropout rate, I don't want to take too difficult ones."
7) "Are there any courses with a low dropout rate, scheduled for Tuesday afternoons, and related to green building?"

Since the preference concepts generated by LLMs are limited, we later test the preference expression generation capability by feeding complex user preference descriptions collected from the downstream interactive dialogue system. The results show that the system can effectively generate simple preference expressions on a one-time basis, such as total credits and dropout rates. However, for nonlinear preference expressions, such as the number of consecutive classes or the number of days with classes, the system may generate code that causes execution errors (e.g., Gurobi.LinExpr error) due to nonlinearity. Additionally, there may be calculation logic errors for different decision variables, resulting in mismatched preference values between the generated solution and the one produced by the reference code. When the current code and error type are provided, the system can effectively refine the code within three iterations.

B. RQ2: Representational Effectiveness of PIR

To assess the extent to which the proposed PIR design can effectively characterize the diverse preference descriptions expressed by users, we conducted a user study using the deployed interactive course timetabling system (as shown in Figure 4). We recruited 12 participants (all second-year undergraduate students) to use the system for planning their actual schedules for the Spring 2025 semester. During the interaction, users were encouraged to express their scheduling requirements naturally via the dialogue interface. This process yielded a corpus of 205 distinct natural language instructions related to personal preferences.

We adopted an *open-coding* approach to analyze this corpus, decomposing each instruction into semantic units and mapped them to the three components of our PIR framework: Condition Types, Preference Expressions (Logic), and Target Items (Scope). The statistical results are shown in Table I.

From the table, we can see that the number of distinct preference expressions is relatively small—only five. However, through the modular design of PIR. these expressions, when combined with different target items, can effectively represent a wide range of semantic concepts, demonstrating the representational effectiveness of PIR. For example, *count_courses* combined with various time slots can express "courses on Monday mornings and Friday afternoons," or, when paired with a list of course names, it can indicate "courses related to Artificial Intelligence." Similarly, *count_consecutive_courses* combined with time slots can represent consecutive courses across adjacent periods, such as "having classes in both the last session of the morning and the first session after lunch." When combined with course or department names, it can express constraints like "no classes scheduled immediately after experimental or physical education courses." Furthermore, the decoupling of condition types allows users to adjust their preferences locally. For instance, the preference for the number of early morning classes can be adjusted from "as few as possible (minimize)" to "at most one (<=1)." This modular approach helps manage the complexity of preference expression and enhances the flexibility of the system.

TABLE I
BREAKDOWN OF USER INSTRUCTIONS INTO PIR COMPONENTS. NUMBERS IN PARENTHESES REPRESENT THE FREQUENCY OF EACH TYPE IDENTIFIED DURING THE OPEN-CODING PROCESS.

Condition Type	Preference Expression	Target Items
Constraints (96)	count_courses (37)	course name (87)
Item Priority (78)	count_consecutive_courses (26)	instructor (84)
Objectives (31)	count_dropout_rate (19)	course time (75)
	count_days (16)	dropout rate (27)
	count_credits (11)	department (21)
	None (96)	credit (19)
		All (37)

TABLE II
PERFORMANCE COMPARISON OF EXACT ACCURACY (%), CONDITION (%), EXPRESSION (%), AND TARGET ITEMS (%) FOR DIFFERENT METHODS.

	Exact Accuracy	Condition	Expression	Target Items
Standard	$29.10_{\pm 2.20}$	$85.20_{\pm 1.23}$	$36.73_{\pm 1.97}$	$47.16_{\pm 1.23}$
Reflexion	$37.89_{\pm 1.71}$	$87.80_{\pm 0.98}$	$41.30_{\pm 1.13}$	$57.56_{\pm 0.98}$
Ours	$93.50_{\pm 0.75}$	$97.89_{\pm 0.56}$	$99.84_{\pm 0.28}$	$96.10_{\pm 0.49}$

C. RQ3: Performance Superiority via PIR

To evaluate the necessity and effectiveness of the proposed PIR layer compared to end-to-end preference modeling, we conducted a comparative evaluation on a corpus of 205 natural language instructions collected from the user study.

We compared our framework against two strong baselines based on GPT-4o:

1) Standard Prompting: Directly generating the solver code with zero examples.
2) Reflexion [14]: An agentic workflow that iteratively refines the generated code through a "generate–execute–reflect–regenerate" loop. The model first generates initial code, then executes it locally to collect

runtime feedback including Python exception traces and solver error messages. This feedback, along with the previous code version, is passed back to the model to summarize the failure and regenerate a corrected version. The maximum number of iterations is set to 3.

As shown in Table II, we report the average results of three independent runs for each method. Our PIR-based natural language parsing achieved a dominant **Exact Accuracy of 93.50%** (std = 0.75), which essentially meets the accuracy requirements for interactive systems. A more detailed analysis reveals that while the baseline method performs well in condition recognition, its accuracy for expressions and items is relatively low. The PIR-based approach, by splitting these components, effectively improves the generation accuracy of individual modules, thereby boosting overall accuracy. Additionally, the reflection mechanism, built upon the standard method, enhances the recognition of target items and expression generation through multiple rounds of reflection, although the overall improvement remains limited.

VI. Case Study: Travel Planning

To explore the applicability of our framework beyond course timetabling, we conduct a pilot case study on travel planning, a domain involving multiple heterogeneous candidate types (attractions, restaurants, transactions, accommodations, points of interest, and intercity_transport). Rather than a comprehensive evaluation, this study aims to illustrate how the framework adapts to a structurally different domain and identify where the pipeline performs well and where human intervention remains necessary. We use the ChinaTravel dataset [15], which contains real-world travel planning tasks across major Chinese cities.

A. Phase 1: Domain Analysis

a) Decision Variables: The system generates a total of 9 decision variables: 7 binary variables, including 5 with multi-dimensional indexing (e.g., attraction and restaurant visits indexed by (attraction, day, time slot) and (restaurant, day, meal slot)), and 2 with single-dimensional indexing for nightly decisions (e.g., accommodation selection by (accommodation, day)) and transport choices (e.g., train or flight, indexed by transport ID). Additionally, there are 2 continuous variables representing arrival time and cumulative expenditure. However, `budget_used` is redundant, as it can be derived from the selected item costs.

b) Global Constraints: The system generates a total of 12 global constraints, covering core feasibility requirements such as *single accommodation per night*, *no time overlap* between activities, *intercity transport uniqueness*, *opening hours compliance*, and *travel time feasibility*. However, the output contains both gaps and misclassifications. On the gap side, the system misses structural ordering constraints on transport: specifically, that departure transport must be the first scheduled event of the trip and return transport must be the last. On the misclassification side, *meal slot coverage*, requiring at least one restaurant per meal slot per day, is generated as a hard feasibility constraint, whereas it is more appropriately treated as a soft user preference, since travelers may choose to skip or combine meals.

c) Preference Concepts: The system generates a total of 25 preference concepts covering a diverse range of optimization dimensions, such as *total trip cost, accommodation rating, train type, cuisine variety across meal slots, activity balance across days, and same-category attractions on consecutive days.* The breadth of the generated concepts reflects the commonsense knowledge of the underlying language model, which can enumerate meaningful user preferences across a complex domain without explicit domain-specific prompting.

B. Phase 2: Item Selector Generation

Without explicit instruction, the system autonomously generates separate `search_target_items` functions for all six candidate types, correctly inferring the multi-type structure from the data schema. A test suite of 147 cases covering categorical, numeric, structured string, free-text, and edge-case attributes is constructed, with 141 passing. Manual inspection reveals that all 6 failures result from incorrect expected values in the test cases, such as failing to account for substring matching of "swimming pool" with "celebrity swimming pool" and "private swimming pool," yielding 44 rows instead of the predicted 39. This highlights a practical benefit of automated test generation: discrepancies between predicted and actual counts uncover latent specification ambiguities that would otherwise go undetected.

C. Phase 3: Preference Expression Generation

From the 25 preference concepts, the system generates a library of 13 scope-agnostic expression functions after deduplication, alongside scope definitions for each concept. Scope extraction is accurate across all 25 concepts. However, some redundancy is observed in the function library: for example, *total_selected_attribute_sum* and *daily_selected_attribute_sum* share identical logic differing only in index scope, suggesting that a more principled design would parameterize the time scope rather than instantiate separate functions.

For verification, all 13 functions execute without error in both the MILP (f_{milp}) and oracle (f_{oracle}) implementations (13/13 executability). Twelve produce numerically matching results (12/13 accuracy). The single failure occurs in *consecutive_same_category_overlap*, where the big-M coefficient is set to the category size rather than 1, loosening the lower-bound constraint to $z \geq 0$ under minimization and allowing the solver to suppress the indicator variable incorrectly. This error is invisible to syntax validators but immediately caught by comparing f_{milp} output (1) against f_{oracle} output (2) on the same assignment, confirming the practical value of dual-mode verification for detecting semantic linearization bugs.

D. Discussion

The travel planning case study is a pilot exploration rather than a comprehensive evaluation, and further study is needed to fully assess the framework's capability in such domains. Nevertheless, the study suggests that the framework has the

potential to handle more complex domains with multiple heterogeneous candidate types, while also revealing that human collaboration plays a more critical role compared to the course timetabling domain, spanning constraint classification in Phase 1, test case validation in Phase 2, and abstraction consolidation in Phase 3.

More broadly, although the system generates a rich set of preference concepts, they are largely derived from the data schema and problem structure. Preferences rooted in personal habits or nuanced attribute combinations may go unelicited, suggesting that this phase benefits from iterative human feedback rather than single-pass generation.

Overall, the framework is most effective as a *collaborative scaffolding tool*: it handles the breadth-requiring and mechanical parts of model construction, while the human collaborator focuses on judgment-intensive and compositional tasks.

VII. Conclusion

In this paper, we tackle the challenges of interactive optimization modeling over large-scale data with evolving user preferences. We introduce the Preference Intermediate Representation (PIR) and propose an automated framework for generating domain-specific PIR components from raw data and task descriptions. Evaluations on university course timetabling demonstrate the effectiveness of our approach, and a pilot case study on travel planning suggests its potential applicability to more complex domains with heterogeneous multi-type candidate structures. Our work offers a scalable foundation for preference modeling in real-world optimization tasks. Future work will focus on more comprehensive evaluations across diverse planning domains.

Acknowledgments

This work was supported by the Science and Technology Innovation Key R&D Program of Chongqing under Grant No. CSTB2023TIAD-STX0033.

References

[1] J. C. Manzano, A. F. O. Soliven, A. M. B. Llamas, S. M. V. Tinsay, B. P. V. Samson, and R. A. Cabredo, "Using boolean satisfiability solvers to help reduce cognitive load and improve decision making when creating common academic schedules," in *CHI '21: CHI Conference on Human Factors in Computing Systems, Virtual Event / Yokohama, Japan, May 8-13, 2021*, Y. Kitamura, A. Quigley, K. Isbister, T. Igarashi, P. Bjørn, and S. M. Drucker, Eds. ACM, 2021, pp. 456:1–456:13. [Online]. Available: https://doi.org/10.1145/3411764.3445681

[2] D. Ju, S. Jiang, A. Cohen, A. Foss, S. Mitts, A. Zharmagambetov, B. Amos, X. Li, J. T. Kao, M. Fazel-Zarandi, and Y. Tian, "To the globe (TTG): towards language-driven guaranteed travel planning," in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: EMNLP 2024 - System Demonstrations, Miami, Florida, USA, November 12-16, 2024*, D. I. H. Farías, T. Hope, and M. Li, Eds. Association for Computational Linguistics, 2024, pp. 240–249. [Online]. Available: https://doi.org/10.18653/v1/2024.emnlp-demo.25

[3] J. Xie, K. Zhang, J. Chen, T. Zhu, R. Lou, Y. Tian, Y. Xiao, and Y. Su, "Travelplanner: A benchmark for real-world planning with language agents," in *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, ser. Proceedings of Machine Learning Research, R. Salakhutdinov, Z. Kolter, K. A. Heller, A. Weller, N. Oliver, J. Scarlett, and F. Berkenkamp, Eds. PMLR / OpenReview.net, 2024, pp. 54 590–54 613. [Online]. Available: https://proceedings.mlr.press/v235/xie24j.html

[4] C. Lawless, J. Schoeffer, L. Le, K. Rowan, S. Sen, C. S. Hill, J. Suh, and B. Sarrafzadeh, ""i want it that way": Enabling interactive decision support using large language models and constraint programming," *ACM Trans. Interact. Intell. Syst.*, vol. 14, no. 3, pp. 22:1–22:33, 2024. [Online]. Available: https://doi.org/10.1145/3685053

[5] R. Ramamonjison, T. T. L. Yu, R. Li, H. Li, G. Carenini, B. Ghaddar, S. He, M. Mostajabdaveh, A. Banitalebi-Dehkordi, Z. Zhou, and Y. Zhang, "Nl4opt competition: Formulating optimization problems based on their natural language descriptions," in *NeurIPS 2022 Competition Track, November 28 - December 9, 2022, Online*, ser. Proceedings of Machine Learning Research, M. Ciccone, G. Stolovitzky, and J. Albrecht, Eds. PMLR, 2021, pp. 189–203. [Online]. Available: https://proceedings.mlr.press/v220/ramamonjison22a.html

[6] Z. Xiao, D. Zhang, Y. Wu, L. Xu, Y. J. Wang, X. Han, X. Fu, T. Zhong, J. Zeng, M. Song, and G. Chen, "Chain-of-experts: When llms meet complex operations research problems," in *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net, 2024. [Online]. Available: https://openreview.net/forum?id=HobyL1B9CZ

[7] C. Huang, Z. Tang, S. Hu, R. Jiang, X. Zheng, D. Ge, B. Wang, and Z. Wang, "ORLM: A customizable framework in training large models for automated optimization modeling," *Oper. Res.*, vol. 73, no. 6, pp. 2986–3009, 2025. [Online]. Available: https://doi.org/10.1287/opre.2024.1233

[8] Y. Hao, Y. Chen, Y. Zhang, and C. Fan, "Large language models can solve real-world planning rigorously with formal verification tools," in *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL 2025 - Volume 1: Long Papers, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, L. Chiruzzo, A. Ritter, and L. Wang, Eds. Association for Computational Linguistics, 2025, pp. 3434–3483. [Online]. Available: https://doi.org/10.18653/v1/2025.naacl-long.176

[9] A. AhmadiTeshnizi, W. Gao, and M. Udell, "Optimus: Scalable optimization modeling with (MI)LP solvers and large language models," in *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, ser. Proceedings of Machine Learning Research, R. Salakhutdinov, Z. Kolter, K. A. Heller, A. Weller, N. Oliver, J. Scarlett, and F. Berkenkamp, Eds. PMLR / OpenReview.net, 2024, pp. 577–596. [Online]. Available: https://proceedings.mlr.press/v235/ahmaditeshnizi24a.html

[10] X. Huang, Q. Shen, Y. Hu, A. Gao, and B. Wang, "Llms for mathematical modeling: Towards bridging the gap between natural and mathematical languages," in *Findings of the Association for Computational Linguistics: NAACL 2025, Albuquerque, New Mexico, USA, April 29 - May 4, 2025*, ser. Findings of ACL, L. Chiruzzo, A. Ritter, and L. Wang, Eds. Association for Computational Linguistics, 2025, pp. 2678–2710. [Online]. Available: https://doi.org/10.18653/v1/2025.findings-naacl.146

[11] H. Chen, G. E. Constante-Flores, and C. Li, "Diagnosing infeasible optimization problems using large language models," *INFOR Inf. Syst. Oper. Res.*, vol. 62, no. 4, pp. 573–587, 2024. [Online]. Available: https://doi.org/10.1080/03155986.2024.2385189

[12] H. Chen, G. E. Constante-Flores, K. S. I. Mantri, S. M. Kompalli, A. S. Ahluwalia, and C. Li, "Optichat: Bridging optimization models and practitioners with large language models," *CoRR*, vol. abs/2501.08406, 2025. [Online]. Available: https://doi.org/10.48550/arXiv.2501.08406

[13] B. Li, K. Mellou, B. Zhang, J. Pathuri, and I. Menache, "Large language models for supply chain optimization," *CoRR*, vol. abs/2307.03875, 2023. [Online]. Available: https://doi.org/10.48550/arXiv.2307.03875

[14] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: language agents with verbal reinforcement learning," in *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., 2023. [Online]. Available: https://dl.acm.org/doi/10.5555/3666122.3666499

[15] J. Shao, X. Yang, B. Zhang, B. Chen, W. Wei, G. Cai, Z. Dong, L. Guo, and Y. Li, "Chinatravel: A real-world benchmark for language agents in chinese travel planning," *CoRR*, vol. abs/2412.13682, 2024. [Online]. Available: https://doi.org/10.48550/arXiv.2412.13682