IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 26 July 2025, accepted 18 August 2025, date of publication 21 August 2025, date of current version 29 August 2025.

Digital Object Identifier 10.1109/ACCESS.2025.3601518

RESEARCH ARTICLE

Improving Adaptability in Optimization-Based Decision Support Systems Through Large Language Models

GIANPAOLO GHIANI, EMANUELE MANNI, AND SANDRO ZACCHINO
Dipartimento di Ingegneria dell'Innovazione, Università del Salento, 73100 Lecce, Italy

Corresponding author: Gianpaolo Ghiani (gianpaolo.ghiani@unisalento.it)

This work was supported in part by the Ministero dell'Università e della Ricerca (MUR) of Italy, Progetto di Rilevante Interesse Nazionale (PRIN) Project "Optimizing Sustainable Multi-Modal and Multi-Tasking Last-Mile Distribution System with Carbon-Free Autonomous Vehicles, Ground Robots, Drones, and Public Transport" under Grant CUP Master H53D23002000006 and Grant CUP F53D23002720006.

ABSTRACT Since the 1950s, optimization-based Decision Support Systems have emerged as one of the most impactful applications of Operations Research, replacing human-based planning across a wide range of industries. These systems have greatly improved decision-making in terms of efficiency, accuracy, and scalability. However, they also come with a significant limitation: a lack of flexibility and adaptability. Typically, the algorithms used in these systems are predefined during the design phase, following a thorough-and often time-consuming-requirements analysis. They are then hard-coded into the system, making them well-suited for specific, well-understood scenarios. However, this rigidity makes it difficult for such systems to adapt to unforeseen changes or evolving operational needs. As a result, expert intervention, re-analysis, or even a complete system redesign may be required-often at considerable time and cost. This paper explores how *Large Language Models* (LLMs) can be integrated with traditional optimization algorithms to address this key limitation-lack of flexibility-while preserving their core strengths: speed and solution quality. The core idea is to leverage LLMs to interpret natural language instructions, reconfigure algorithmic components, support preference-based decision-making, and explain the rationale behind their choices. Computational results on a "rich" *Vehicle Routing Problem* (VRP) setting-a class of VRPs that incorporate multiple real-world constraints and complexities commonly encountered in last-mile distribution-demonstrate the potential of this hybrid approach.

INDEX TERMS Decision support systems, optimization, large language models, vehicle routing.

I. INTRODUCTION

Optimization-based Decision Support Systems (DSSs) are systems that leverage mathematical models and optimization techniques to aid in decision-making, particularly in complex environments where resources are limited and multiple objectives must be balanced ([1], [2]). As some of the most impactful applications of Operations Research (OR), they serve as critical tools across a wide range of industries - including manufacturing, transportation, logistics, and e-commerce. In today's fast-paced, resource-constrained markets, these systems have become indispensable for enhancing productivity and fostering strategic advantages.

Although the transition from manual to algorithmic planning has proven successful, optimization-based DSSs come with a major limitation: a lack of flexibility and adaptability. Typically, the algorithms are defined during the initial design phase, based on a thorough and often lengthy analysis of system requirements. Once implemented, they are embedded directly into the system's code. As a result, these systems are optimized for specific, anticipated scenarios but often struggle to handle unexpected situations or changing requirements during operation. When such challenges arise, expert intervention is usually needed - this could involve revisiting the original analysis or even redesigning

The associate editor coordinating the review of this manuscript and approving it for publication was Chi-Tsun Cheng.

TABLE 1. Comparison of planning approaches.

Feature	Manual planning	Planning with traditional optimization-based DSSs	Planning with our LLM-optimization-based framework
Flexibility	High (human adapts plans)	Low (fixed algorithms)	High (LLM reconfigures algorithmic components based on instructions)
Scalability	Low	High	High
Interaction mode	Fully manual	Through a Graphical User Interface (GUI)	Natural language input/output (possibly in conjunction with a GUI)
Responsiveness to preferences	High (subjective human judgment)	Low (requires reprogramming)	High (preferences parsed and incorporated by LLM)
Explainability	High (human rationale)	Low (black-box behavior)	High (LLM provides rationale in natural language)
Computational efficiency	Low	High	High (with minimal LLM overhead)
Human-in-the-loop validation	Always required	Optional / not typical	Built-in fallback mechanisms for safety
Adaptability to new contexts	Medium (relies on human expertise)	Low (requires system redesign)	High (language-based guidance enables reconfiguration)

significant parts of the system. These adjustments can be both time-intensive and expensive. These limitations become particularly problematic in dynamic, fast-paced environments with frequent disruptions or evolving constraints - a defining trait of an increasingly large number of sectors in the modern economy.

Several studies have examined the abandonment or failure of optimization-based DSSs, often attributing these outcomes to shifts in environmental or organizational conditions. For instance, [3] presents a case study in which, shortly after deployment, managers began modifying the optimal solutions generated by a DSS in response to a significant environmental change. These continual adjustments ultimately led to the system's discontinuation within just six months. Similarly, [4] underscores that the inherent volatility of task environments, coupled with the complex and unpredictable nature of managerial decision-making, increases the likelihood of failure in DSS implementations. These findings highlight the critical importance of designing DSSs that are adaptable to evolving contexts and aligned with the practical realities of organizational decision-making.

This paper investigates how *Large Language Models* (LLMs) can be integrated with traditional optimization algorithms to overcome a key limitation - lack of flexibility - while preserving their fundamental strengths, such as speed and solution quality. The core idea is to

- leverage LLMs to interpret *natural language* instructions,
- dynamically reconfigure algorithmic components based on evolving requirements,
- support preference-based decision-making, and
- provide transparent explanations for the rationale behind the generated solutions.

Table 1 presents a comparative overview of manual planning, traditional algorithmic planning typically employed by conventional optimization-based DSSs, and the hybrid approach introduced in this study. The table highlights key differences in terms of methodology, efficiency, and adaptability, as discussed in greater detail in the following sections of the paper.

The advantages of our approach lie in its ability to enhance the adaptability and flexibility of traditional optimization algorithms, enabling them to dynamically respond to evolving operational requirements without the need for extensive system redesign. By leveraging the natural language understanding capabilities of LLMs, the system can interpret user intent, reconfigure algorithmic parameters on-the-fly, and provide transparent, human-understandable explanations for its decisions. This not only improves responsiveness in complex, real-world decision-making scenarios but also reduces reliance on domain experts for frequent reprogramming or manual intervention.

The remainder of the paper is organized as follows. In Section II, we review the related literature, highlighting key studies and foundational concepts relevant to our work. In Section III, we describe our approach in detail, outlining the methodology and the underlying framework. To demonstrate the effectiveness of this hybrid approach, we conduct an extensive computational study on a *Vehicle Routing Problem* setting ([5], [6]), commonly encountered in last-mile distribution, highlighting potential improvements in flexibility and user engagement. In this respect, Section IV presents a comprehensive evaluation of our framework, including the results from our computational experiments and their implications. Finally, in Section V, we summarize our findings, discuss the contributions of our work, and suggest potential avenues for future research.

II. RELEVANT LITERATURE

In this section, we present a comprehensive review of the emerging literature on the integration of optimization algorithms and LLMs within decision support systems. This interdisciplinary research area, situated at the intersection of *Natural Language Processing* and *Operations Research*, has garnered increasing attention in recent years, driven by rapid advancements in LLM capabilities.

Large Language Models are advanced machine learning systems trained on vast amounts of textual data, enabling them to understand and generate human language with a high degree of fluency [7]. These models have demonstrated significant capabilities in tasks such as natural language understanding, text generation, and semantic reasoning, making them valuable tools across a variety of domains. A key feature of interacting with LLMs is the use of *prompts* - specific inputs or instructions in natural language - given to the model to guide its response [8]. Prompts can be classified into two main types: instructional prompts, which provide explicit guidance on the desired output, and contextual prompts, which supply background information or examples to help the model generate more relevant or accurate responses. *Zero-shot prompting* involves asking the model to perform a task without providing any specific training examples, relying solely on its pre-existing knowledge to generate the correct output. On the other hand, *few-shot prompting* is a technique that provides a limited number of examples to help the model better grasp the task and generate more accurate results. As LLMs scale in size and complexity, they exhibit ''emerging properties'' - unexpected behaviors and advanced competencies that arise from their intricate architectures, which were not explicitly programmed but emerge through the training process [9].

For clarity and conciseness, we now categorize the literature into two main subfields: (1) the generation of optimization models and (2) the development of heuristic algorithms, both derived from textual descriptions.

A. GENERATING LINEAR PROGRAMMING FORMULATIONS FROM TEXTUAL DESCRIPTIONS

This subfield is centered on generating optimization problem formulations from text descriptions, with the goal of making optimization solvers more accessible to non-experts through natural language. Successfully accomplishing this task requires substantial domain expertise and familiarity with optimization modeling and associated solvers. Additionally, an optimization problem can often be expressed in multiple valid ways, adding complexity to the process.

Research in this area is closely related to *automatically solving math word problems* (MWPs), a field that dates back to the 1960s. An MWP is presented in a narrative or story-like format and requires mathematical concepts or operations to solve. The goal is to translate the information in the text into a mathematical equation or model and then apply appropriate mathematical methods to find the solution. Early approaches, such as those in [10], explored mapping natural language text to predefined mathematical templates. In contrast, more recent methods rely on deep-learning techniques, as reviewed in [11] and [12].

It is also closely related to the field of *semantic parsing for formal languages*, which involves the process of converting natural language utterances into logical forms or representations that can be directly interpreted and executed by computational systems. These logical forms are often structured in formats such as SQL, SPARQL, formal logic, or domain-specific languages (DSLs). The key objective is to enable machines to understand human language in a way that allows them to interact effectively with knowledge bases, databases, or other structured systems. For an in-depth overview, refer to the recent surveys by [13] and [14].

The research on automatically generating optimization models was stimulated by the *NL4Opt Competition* held in 2022 [15], which focused on developing *Linear Programming* (LP) formulations from textual descriptions. The competition comprised two sub-tasks: (1) recognizing and labeling the semantic entities of optimization problems to reduce ambiguity, and (2) generating an LP formulation of the optimization problem that could be directly used by commercial solvers. Reference [16] describes the dataset used in the contest and compares the performance of the ChatGPT [17] large language model against the winning solutions. Reference [18] evaluates the performance of various LLMs, including GPT-3.5, GPT-4, and Llama-2-7b, in translating natural language descriptions into mathematical optimization models. It also introduces LM4OPT, a fine-tuning framework for Llama-2-7b, and highlights the gap in contextual understanding between smaller and larger models, setting the stage for future advancements in this field. Reference [19] introduces an automated approach to generating mathematical optimization models from natural language descriptions, addressing key challenges in defining, searching, and evaluating optimization formulations. By leveraging LLMs within a Monte-Carlo tree search framework, the method improves formulation generation and correctness evaluation, demonstrating superior performance and efficiency in benchmarks for linear and mixed-integer programming problems. Reference [20] presents a multi-agent, multi-stage framework for translating natural language descriptions into optimization models, using agents for relation identification and verification without relying on solver execution. Through extensive experiments, the study demonstrates that the proposed framework outperforms traditional LLM prompting techniques in optimization modeling tasks. Reference [21] proposes a ''decision optimization copilot'', an AI assistant that interacts with decision makers in natural language to understand business problems, automatically formulate the corresponding optimization models, and solve them using appropriate algorithms. Reference [22] summarizes existing works, including the datasets they propose, the size of each dataset, and the types of optimization problems they support (*LP*, *Mixed LP*, *Mixed Integer LP* and *Non-linear Programming*). They also introduce a framework based on four essential roles: (1) a *formulator* that translates natural language problem descriptions into precise mathematical formulations; (2) a *planner* that develops a high-level solution strategy before execution; and (3) both a *coder* and a *code critic* that interact with the environment and assess outcomes to improve future actions. Experiments demonstrate that all roles are crucial, as removing the planner

or code critic results in productivity drops of 5.8× and 3.1×, respectively. Specialized versions of the above approach have recently been proposed by various authors, including [23] which presents a method for automatically formulating and solving *stochastic* optimization problems from natural language descriptions. In the context of *Vehicle Routing Problems*, [24] investigates whether LLMs can model robotic applications. Using a dataset with eight variants, the study has tested three different LLMs. The results have showed that the models' performance is sensitive to the context provided, with performance varying depending on whether the prompts included relevant information, such as related papers and pseudocode. The study concludes by proposing two directions for enhancing LLMs performance in routing tasks.

B. HEURISTIC GENERATION FROM TEXTUAL DESCRIPTIONS

Building on the observation that many planning problems are too complex to be effectively addressed by model-based approaches relying on general-purpose solvers, this line of research seeks to develop optimization heuristics or metaheuristic algorithms directly from textual descriptions. The aim is to make optimization techniques more accessible to non-experts through natural language.

This line of research is a subfield of *program synthesis from natural language*, which focuses on translating requirement specifications into executable code. Unlike traditional programming, where humans write code manually, this approach seeks to automate the process by enabling machines to generate functional code directly from natural language inputs. The goal is to bridge the gap between human intuition, expressed in natural language, and the precision required for writing computer programs. While the emphasis is not necessarily on optimization, the generated code should still be correct, efficient, and functional based on the given description. Applications span a wide range of fields, from software development and data analysis to educational tools that help users learn programming without needing to master a specific language. Additionally, this research can be applied to automate repetitive coding tasks, assist with debugging, and enhance productivity in programming environments. It holds promise for improving accessibility in tech-related fields by enabling individuals with no formal coding background to create and interact with software more easily. See [25] for a systematic review of the past decade of research in the field, and [26] for a comparative study on the use of LLMs in Python code generation.

Heuristic algorithms have long been a preferred approach for solving *Combinatorial Optimization Problems* (COPs), including VRPs [27]. The development of heuristics for specific COPs has been an active area of research since the early 2000s. Early *Heuristic Generation* (HG) methods typically relied on frameworks composed of predefined modules, balancing exploration and exploitation, but often resulted in limited performance [28]. Only recently has the literature begun to explore the use of large language models (LLMs) for heuristic generation. In this line of research, [29] propose two methods in which LLMs are employed to identify alternative search directions within a baseline Genetic Algorithm [30]. These alternatives are then evaluated across multiple COP instances to identify the one yielding the highest fitness values. In a similar vein, [31] consider *Large Neighborhood Search* [32], a metaheuristic in which, at each iteration, part of the current solution is destroyed (e.g., removing customers from routes), and then repaired using a heuristic or optimization method. They propose using LLMs to generate alternative destroy-and-repair operators, guided by instructions expressed in natural language. Recently, [33] presented a systematic review of over 100 publications exploring the application of LLMs in combinatorial optimization, categorized by the tasks performed by the models, their architectures, the datasets used, and the domains of application.

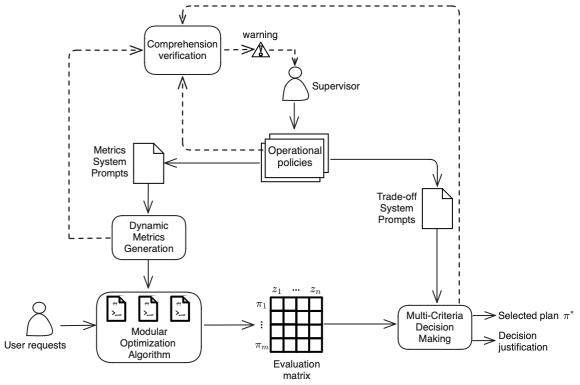

FIGURE 1. Architecture of the proposed framework.

III. FRAMEWORK

As discussed in Section I, our goal is to leverage Large Language Models to address a key limitation of optimization-based Decision Support Systems: their lack of flexibility. While such systems are highly effective in terms of speed and solution quality, they often struggle to accommodate the diverse and evolving demands of real-world decision-making. By integrating LLMs, we aim to enhance the adaptability of these systems without compromising their core strengths.

The central idea is to utilize LLMs to perform the following tasks:

(a) interpret natural language instructions, allowing the *system supervisor* to interact with the DSS in a more intuitive manner; these instructions aim to modify the base policy followed by the DSS, either by changing the metrics used to evaluate decisions or by adjusting the trade-off between different performance measures.
(b) enable dynamic reconfiguration of algorithmic components, allowing the system to adapt to varying problem requirements and constraints;

(c) support preference-based decision-making, ensuring that user preferences and priorities are integrated into the optimization process; and
(d) explain the rationale behind the system's choices, providing transparency and fostering trust in the decision-making process.

This integration seeks to bridge the gap between complex optimization models and user-friendly interaction. In the reminder of this section, we provide an overview of the main components of our framework, as illustrated in Fig.1, and explain how each component contributes to achieving these objectives.

A. DYNAMIC METRICS GENERATION

At the heart of the Decision Support System is an *optimization module* responsible for generating a diverse set of m alternative plans. This module operates within a largely hard-coded algorithmic framework, ensuring consistency and efficiency in core planning processes. However, certain components within this framework - particularly those requiring adaptability or context-sensitive logic - are not predefined. Instead, the responsibility for generating the code for these dynamic procedures is assigned to a specialized component known as the *Dynamic Metrics Generation* (DMG) module. The DMG module plays a crucial role in enabling the system's flexibility and responsiveness to varying decision contexts. It functions under the guidance of a curated set of supervisory inputs known as *metrics system prompts*. These prompts are provided by the (human) *system supervisor* and serve to steer the DMG module in tailoring specific procedures that reflect real-time performance indicators, user preferences, or operational constraints. In this way, the DSS maintains a robust yet adaptable architecture, blending fixed algorithmic structure with dynamically generated components to optimize decision-making across a wide range of scenarios.

B. MULTI-CRITERIA DECISION MAKING

The m alternative plans generated by the optimization algorithm, denoted as $\pi_1, \ldots, \pi_m$, are characterized by n distinct features (or evaluation criteria), represented as $z_1, \ldots, z_n$. These features capture various dimensions of the plans - such as cost, efficiency, risk, or environmental impact - and serve as the foundational data for a *Multi-Criteria Decision Making (MCDM)* module. The MCDM module is responsible for evaluating and ranking the alternative plans based on how well they align with the decision-maker's strategic objectives and subjective preferences. These preferences are communicated through a distinct set of supervisory inputs known as *trade-off system prompts*. These prompts articulate the relative importance of different criteria, encode acceptable compromises, and reflect the supervisor's high-level priorities or policy constraints. By integrating this preference information with the feature data of the plans, the MCDM module applies a structured decision-making methodology, such as weighted scoring, utility theory, or outranking techniques, to identify a *preferred plan*, denoted as $\pi^* \in \{\pi_1, \ldots, \pi_m\}$. This selected plan represents the most desirable option according to the current trade-off landscape, offering a well-balanced solution that best satisfies the defined criteria and supervisory intent.

C. THE COMPREHENSION VERIFICATION PHASE

A distinguishing characteristic of both the DMG and MCDM modules is their reliance on a Large Language Model. As recalled in Section I, LLMs are powerful machine learning models able to understand and generate human language with a high degree of fluency. On the minus side, LLMs operate as stochastic, black-box systems, meaning their outputs are probabilistic rather than deterministic, and the internal decision-making processes are not fully transparent or interpretable. Moreover, when providing instructions to an LLM, the process differs fundamentally from communicating with a human, who can offer immediate acknowledgment and confirmation of understanding. Hence, to enhance reliability and ensure that the modules execute appropriate actions and decisions, both the DMG and MCDM modules are asked to undergo a *comprehension verification phase* aimed to confirm the correct interpretation of the supervisor's directives, as conveyed through the system prompts. For the DMG module, the process proceeds as follows:

(a) if the DMG module succeeds to generate the executable code for the new metric, it is first validated by an LLM (with an appropriate prompt) to verify that it corresponds to the description provided in the prompt;
(b) if this phase is passed, the program is executed on a number of representative examples (sample plans) and the results are compared to the expected metric values;
(c) if the validation phase fails, or the code generates run-time errors, or the results do not match the expected values, the DMG module modifies the generated code based on the errors identified in the previous iteration (a process known as *self-debugging*). The procedure is repeated for at most `max_attempts` iterations.

Should the procedure fail, the system issues a warning to the supervisor, enabling the implementation of appropriate corrective measures. Such failures may result from imprecise or incomplete prompts provided by the supervisor.

Algorithm 1 provides a formal description of the DMG process, including verification of comprehension. The procedure takes as input a metric prompt and an integer variable, `max_attempts`, which specifies the maximum number of attempts to generate code. The prompt is an object containing three fields: `metric`, `examples`, and `expected_results`. These fields correspond to the following: `metric` provides the description of the metric, `examples` contains the test cases to be run on the generated code, and `expected_results` defines the correct outputs if the code is accurate.

Algorithm 1 DMG With Comprehension Verification

```
function DMG_with_verification(prompt, max_attempts)
    attempts ← 0
    code ← ""
    error ← ""
    while attempts ≤ max_attempts do
        code ← generate(prompt.metric, code, error)
        validation_result ← validate(prompt.metric, code)
        if validate_result.outcome = "Success" then
            execution_results ← execute(code, prompt.examples)
            if execution_results = prompt.expected_results then
                return ("Success", code)
            else
                error = execution_results.error
                attempts ← attempts + 1
            end if
        else
            error = validate_result.error
            attempts ← attempts + 1
        end if
    end while
    return ("Failure", error)
end function
```

D. DECISION JUSTIFICATION

Finally, the validation of the MCDM module relies on the LLM's ability to generate a justification for its selection of the final plan, based on a given evaluation matrix and trade-off prompt. This justification outlines how various criteria were evaluated, weighted, and balanced against each other. It explains the rationale behind prioritizing certain options over others, including which trade-offs were deemed acceptable, which constraints were considered most critical, and how conflicting objectives were resolved. By making the decision process transparent, the MCDM module not only supports debugging, but also enhances trust in the system's output.

IV. COMPUTATIONAL EXPERIMENTS

As anticipated in Section I, to evaluate the practical viability of integrating LLMs with traditional optimization-based DSSs, we conducted a computational study focused on a "rich" Vehicle Routing Problem (VRP) [34], a type of VRP that includes multiple real-world constraints and complexities, going beyond the basic or classical VRP. In the classical VRP, the goal is simple: find the optimal set of routes for a fleet of vehicles to deliver goods to a set of customers, minimizing total distance or cost. A "rich" VRP, however, includes extra constraints and features that make the problem much more realistic and harder to solve, offering a well-defined, yet complex, testbed characterized by multiple interacting constraints and evolving priorities.

This case study serves to test the central hypothesis of our work: that LLMs, when combined with classical optimization methods, can introduce meaningful flexibility into decision-making systems without sacrificing solution quality or computational efficiency. In doing so, we aim to demonstrate that this hybrid approach can reduce the cognitive and technical overhead traditionally associated with modifying optimization systems, while also improving responsiveness to contextual shifts. Specifically, our study aims to evaluate the following aspects: the correctness of dynamic metrics generation, multi-criteria decision-making, robustness to ill-defined metrics, robustness to contradictions in MCDM, computational efficiency under varying loads, and interpretability of decisions.

The remainder of this section presents the experimental setup, including the datasets used, and performance metrics considered, followed by a detailed analysis of the results.

A. THE EXPERIMENTAL SET-UP

The "rich" VRP we used to test the core concept presented in this paper is a *Multi-Period Dynamic Vehicle Routing Problem with Pick-ups and Deliveries* incorporating a range of real-world constraints, multiple objectives, and additional complexities that extend beyond the canonical formulation.

The basic version of the problem (see [35], [36], [37], [38], and [39]) - is defined as follows. A fleet of vehicles, operating from a central depot, is tasked with fulfilling pickup and delivery requests that arrive dynamically over time. Each customer request is characterized by various attributes, such as a priority level (e.g., normal or high) and, optionally, time-related preferences for pickup and/or delivery (i.e., time windows). Upon receiving a request, the company must decide whether to accept it in its current form. If accepted, the request is scheduled - assigned to a vehicle, a service day, and a time window - and sequenced, meaning it is placed at a specific position within the vehicle's route for the designated day. Given the fleet's limited capacity, accepting a request may preclude the accommodation of future requests.

However, real-world applications of the problem typically involve a range of additional complexities that go far beyond the basic formulation. In practice, decision-makers must balance multiple, often conflicting objectives and operational constraints. For instance, while the overarching goal is to maximize the number of accepted requests, companies may also prefer to service new requests as early as possible (e.g., today rather than tomorrow), even if that implies longer detours or increased route costs. Alternatively, early servicing may be encouraged only if the detour remains within a tolerable distance, reflecting a trade-off between responsiveness and operational efficiency. Moreover, certain business strategies may involve intentionally rejecting tightly constrained requests - such as those with narrow or fixed time windows - to avoid overconstraining the overall fleet

schedule and preserve flexibility for future arrivals. In dense urban areas, geographical constraints also play a role: for example, requests in congested zones might be avoided during peak hours to mitigate delays. Additional criteria include workload balancing across the fleet to prevent under- or overutilization of specific vehicles, and capacity reservation for high-priority requests expected later in the day, based on historical demand patterns. Other real-world considerations may involve minimizing vehicle returns after a certain hour (e.g., 6:00 p.m.) to reduce overtime costs and respect labor regulations, limiting the assignment of long routes on consecutive days to prevent driver fatigue, and promoting customer-driver continuity, where the same driver is repeatedly assigned to a given customer to foster familiarity and trust.

To perform our study, we developed a prototype implementation using a simple cheapest-insertion heuristic for the Multi-Period Dynamic Vehicle Routing Problem with Pick-ups and Deliveries, ChatGTP-4o [17] as an LLM, and LangChain 0.3 [40] as a framework to facilitate interaction between the LLMs and the other applications. In our tests, the VRP heuristic returns m alternative solutions, each evaluated w.r.t. the customer-defined metrics.

B. PERFORMANCE EVALUATION

Evaluating the performance of our innovative optimization-bases DSS requires a multi-faceted approach that takes into account various dimensions of its effectiveness. Below are key evaluation criteria:

- C_1 Correctness of dynamic metrics generation - Verify that the DMG module generates code snippets that correctly evaluate metrics $z_1, \ldots, z_n$;
- C_2 Correctness of multi-criteria decision-making - Verify that the MCDM module selects plans π^* according to the trade-off prompts;
- C_3 Robustness w.r.t. ill-defined metrics - Assess how the DMG module handles ill-defined metrics prompts;
- C_4 Robustness w.r.t. multi-criteria decision-making - Assess how the MCDM module is able to detect contraddictions and inconsistencies in company policies;
- C_5 Computational efficiency - Evaluate computing times under different levels of request load, fleet size, and metrics complexity;
- C_6 Interpretability of decisions - Assess how well the MCDM module justifies selected plans based on computed metrics.

We now describe the tests to assess our methodology according to criteria C_1-C_6.

1) TEST C_1

Tests C_1 and C_2 aim to evaluate the system's ability to adjust dispatching decisions when company policies change over time. In particular, test C_1 aims to evaluate the correctness of dynamic metrics generation. To this purpose, we input the system with a set of metrics prompts, *all clearly articulated*.

TABLE 2. Description for metrics prompts used in the "rich" VRP experimentation.

Metric	Metric description
z_1	Fraction of accepted requests under the current plan
z_2	Fraction of accepted requests on the current day (i.e., "today") under the current plan
z_3	Total distance traveled by the fleet
z_4	Total duration of vehicle routes (serving as a measure of how actively the vehicles were utilized)
z_5	Total waiting time of vehicle routes (working time during which the vehicles were idle)
z_6	Mean absolute deviation (MAD) of route durations on the current day (i.e., "today") (to achieve a balanced allocation of workload among vehicles)
z_7	Percentage deviation of average route durations between "today" and "tomorrow" (to ensure an even distribution of workload across consecutive workdays)
z_8	Number of customers in a specific zip code visited at specific times (i.e., during peak times)
z_9	Weighted score based on accepted requests over multiple days (i.e., scores of 10, 5, and 0 are assigned to requests scheduled for today, scheduled for a future date, or rejected, respectively.)
z_{10}	Weighted score based on priority vs non-priority accepted requests (i.e., scores of 10 and 5 are assigned to scheduled priority and non-priority requests, respectively)

Each of them contains a description of a metric as well as three examples to be used to test the correctness of the code generated by the DMG module (Line 9 of Algorithm 1). Three categories of prompts are considered:

- purely textual descriptions (*T-prompts*);
- descriptions in terms of both text and formulae (*TF-prompts*);
- descriptions including text, formulae and pseudocodes (*TFP-prompts*).

An example of T-prompt is "Introduce a new metric to evaluate the balance of today's fleet plan, based on the coefficient of variation of the durations of the vehicle routes scheduled for the day." An equivalent TF-prompt might be "Introduce a new metric to assess the balance of today's fleet plan. Let $d_1, …, d_k$ denote the durations of the k vehicle routes in today's plan. The metric is defined as the coefficient of variation (CV) of $d_1, …, d_k$, given by the formula …", where the mathematical formulae were supplied to the LLM in *LaTeX* format. Similarly, an equivalent TFP-prompt might contain a pseudocode of a step-by-step procedure for calculating CV. It is worth noting that both the formulae and the pseudocodes are expressed in LaTeX.

We generated the metrics prompts as follows. We began by defining a set of seed metrics (see Table 2 for the purely textual versions). For each metric, we used an LLM to generate multiple paraphrased variations, resulting in a test set of 30 distinct prompts for each prompt type.

TABLE 3. Accuracy of the DMG module for different types of metrics prompts.

Type	Generated	Avg Attempts	Tests Passed
T-prompts	27	1.42	27
TF-prompts	25	1.58	25
TFP-prompts	27	1.42	27

For each prompt, the DMG module generates a corresponding code snippet, which is then tested according to the comprehension verification procedure described in Section III-C. The results of these evaluations are reported in Table 3. Computational results show that the DMG module was able to generate correct code for each T-prompt, except for those related to z_5 (total waiting time). This difficulty arises because deriving the formulae to compute such a performance measure is not straightforward, even for an educated human being. Specifically, the relevant formulae are: $a_j = d_i + t_{ij}$, $s_j = \max(e_j, a_j)$, $w_j = \max(0, e_j - a_j)$, $d_j = s_j$, where d_i is the departure time from the previous customer i, t_{ij} is the travel time from i to j, $[e_i, l_i]$ is the time window of customer i, and a_j, s_j, d_j are the arrival time, the service time, and the departure time from customer j, respectively. On the other hand, when this formula (or an equivalent pseudocode) was provided, the DMG module was able to generate the proper code. This suggests that adding a formula or pseudocode to a metric's description may help the LLM succeed. Unfortunately, experiments showed that this is not always the case: when complex formulae and pseudocode were input, the LLM may struggle to interpret them, resulting in 5 failures for TF-prompts and 3 failures for TP-prompts, respectively. Notably, these failures occurred for performance measures where purely textual prompts succeeded. This may indicate that, in a real-world setting, multiple attempts may be necessary for the system supervisor to find an appropriate balance between the simplicity of T-prompts and the precision of TF- and TP-prompts.

Finally, Table 3 demonstrates the usefulness of the DMG module's self-debugging feature, as an average of about 1.5 attempts were needed to produce correct metrics evaluation code. The results also show that once a code snippet has been validated, it always passes the tests associated with the prompts.

2) TEST C_2

This test aims to verify that the MCDM module selects plan π^* according to the trade-off prompts. To this purpose, we first created a set of seed trade-off prompts (see Table 4). Then for each such prompt, we used an LLM to generate multiple prompt variations through paraphrasing, resulting in a test set of 30 distinct prompts. For each prompt, the MCDM module was asked to generate the corresponding preferred plan π^* for evaluation matrices of different sizes (m = 10, 50, 100 and $n = 10$). The results of these evaluations are reported in Table 5.

TABLE 4. Short topic descriptions for various trade-off prompts.

Trade-off prompt #	Trade-off description
1	Maximize the number of accepted requests. Additionally, prioritize servicing new requests as soon as possible (e.g., today rather than tomorrow), even if it results in longer tours.
2	Maximize the number of accepted requests. Furthermore, service new requests as soon as possible (e.g., today rather than tomorrow), but only if this does not result in longer tours.
3	Maximize the number of accepted requests but reject tight time-window requests (e.g., requests requiring service at an exact time); this criterion aims to prevent overconstraining the fleet plan, which could hinder the acceptance of additional requests later.
5	Maximize the number of accepted requests and prioritize servicing new requests as soon as possible (e.g., today rather than tomorrow). However, avoid servicing requests located in specific ZIP code zones (e.g., 00184, covering areas like the Colosseum, Roman Forum, and Monti in central Rome) if scheduled during peak hours (e.g., 11:30 a.m. – 1:30 p.m.), to reduce the risk of delays due to local congestion.
6	Maximize the number of accepted requests and prioritize servicing new requests as soon as possible (e.g., today rather than tomorrow). However, maintaining route balance is essential—avoid overloading some vehicles while underutilizing others.
7	Maximize the number of accepted requests, but reserve a portion of vehicle capacity for high-priority requests expected later in the day, based on historical patterns.
8	Maximize the number of accepted requests while minimizing the number of vehicles returning to the depot after a specified time (e.g., 6:00 p.m.), to reduce overtime costs and respect driver schedules.
9	Maximize the number of accepted requests while ensuring that no vehicle is assigned long routes on consecutive days, to prevent driver fatigue and promote fair workload distribution.
10	Maximize accepted requests while promoting customer familiarity by minimizing the number of different drivers assigned to the same customer over a week.

Computational results show that when a small number ($m = 10$) of alternative plans are evaluated by the MCDM module, all the selected plans π^* align with the trade-offs and preferences specified in the prompt. As m increases, the accuracy slightly decreases. However, a closer examination of the results reveals that even when π^* is not the optimal trade-off among the performance measures, it remains close to the optimum and provides an excellent practical solution.

Another important aspect of the MCDM procedure is that if two or more plans are tied with respect to the performance measures described in the prompt, the LLM selects the plan that performs best according to common-sense reasoning applied to the other measures. An example of this behavior is discussed in Section IV-B6.

TABLE 5. Accuracy of the MCDM module for different types of trade-off prompts.

Number of plans m	Correct
10	30
50	28
100	25

TABLE 6. Performance of the DMG module by type of metrics prompts and degradation level.

Type of metrics prompts	Informal		Unclear		Ambiguous	
	Tests		Tests		Tests	
	Generated	Passed	Generated	Passed	Generated	Passed
T-prompts	21	21	18	15	18	15
TF-prompts	18	18	18	18	21	21
TFP-prompts	24	24	24	24	24	24

TABLE 7. Impact of trade-off prompt clarity on the MCDM phase.

Degradation level	Correct
1	30
2	30
3	24

TABLE 8. Average token usage and computing times per task.

Task		Token/run	Time (in sec)
DMG	Code generation	7,114	3.75
	Validation	6,254	0.99
MCDM	$m = 10$	1,713	4.27
	$m = 50$	5,537	6.12
	$m = 100$	10,107	7.16
	$m = 200$	21,882	10.27
	$m = 500$	54,772	12.42
	$m = 1000$	109,736	15.09

3) TEST C_3

Test C_3 aims to evaluate the robustness of the proposed approach with respect to ill-defined metrics. To this end, an LLM was used to randomly modify each of the metric prompts from test C_1, gradually decreasing their clarity and coherence. Specifically, the LLM was instructed to make the C_1 prompts moderately "informal," "unclear," and "ambiguous," respectively. For TF- and TP-prompts, only the textual parts were altered by these modifications.

For each modified prompt, the DMG module generated the corresponding code snippet. The evaluation results (Table 6) show a consistent degradation in performance, demonstrating that clearly articulated prompts (like those in test C_1) are essential for correct code generation. In particular, the results indicate that the LLM can handle informal language to some extent but struggles with unclear or ambiguous prompts. Furthermore, the presence of formulae and especially pseudocode helps mitigate inaccuracies arising from imperfectly expressed prompts.

4) TEST C_4

Test C_4 aims to evaluate how the clarity of trade-off prompts affects the quality of the MCDM phase. To this end, an LLM was used to progressively reduce the clarity of the prompts from test C_2. The results are shown in Table 7, where the degradation levels are labeled 1, 2, and 3. The findings indicate that the MCDM module is generally robust to prompt clarity: only at degradation level 3 does it make six errors out of 30 prompts when selecting plan π^*.

5) TEST C_5

LLMs excel at understanding and generating natural language but can be relatively slow due to their inherently sequential processing, where each token is generated in order. This limits parallelization and increases latency during inference. In our system, the most critical phase is the MCDM step, which must be performed each time a new request arrives, as the system needs to update the current plan. In contrast, the DMG phase is less critical, as it is only executed when the decision-making policy requires updating. With that in mind, test C_5 is designed to evaluate the token usage and computational efficiency of the DMG and MCDM modules. As shown in Table 8, the DMG module performs extremely efficiently in both the generation and validation phases, making it well-suited for real-world applications. In contrast, the performance of the MCDM phase depends on the number of alternative plans m considered when selecting π^*. The results indicate that the procedure scales well, as both token usage and computation time increase gradually as m ranges from 10 to 1000 candidate plans.

6) TEST C_6

Test C_6 aims to evaluate the interpretability of the decisions made by the MCDM module. To this end, we reused the evaluation matrices and trade-off prompts from test C_2 to assess how effectively the MCDM module justified the selected plans based on the computed metrics. According to our qualitative evaluation, all justifications provided by the MCDM module in the C_2 tests were satisfactory. As an example, we present the explanation given for selecting plan π_3, using the evaluation matrix in Table 9 and the third trade-off prompt from Table 4: *"Plan π_3 is selected as the best solution because it aligns well with the company policy of maximizing accepted requests and servicing new requests as soon as possible. It has one of the highest fractions of accepted requests ($z_1 = 0.98$) and accepts all requests for today ($z_2 = 1.0$). The total distance traveled ($z_3 = 2190.33$) is relatively low, which suggests efficient routing without excessive detours. Additionally, it has a low number*

of customers visited during peak times ($z_8 = 4$), which is favorable. The weighted score for accepted requests over multiple days ($z_8 = 235$) and the weighted score for priority vs nonpriority requests ($z_9 = 305$) are also strong, indicating a good balance between immediate and future scheduling. Overall, Plan π_3 provides a balanced approach to meeting the company's objectives while maintaining operational efficiency." It is worth noting that this explanation reveals that when two or more plans are equally ranked based on the performance measures specified in the prompt, the LLM defaults to selecting the plan that performs better according to common-sense reasoning applied to the remaining criteria. This behavior suggests a degree of reasoning flexibility that may enhance the quality of the decision-making process.

TABLE 9. Example of evaluation matrix generated by the VRP heuristic w.r.t. the metrics $z_1, \ldots, z_{10}$ described in Table 2.

Plan	z_1	z_2	z_3	z_4	z_5	z_6	z_7	z_8	z_9	z_{10}
π_1	0.98	1.00	2352	4248	526	18	2	7	230	305
π_2	0.97	1.00	2430	4409	502	232	34	5	225	315
π_3	0.98	1.00	2190	4294	513	140	7	4	235	305
π_4	0.97	1.00	2115	3759	493	47	15	11	230	315
π_5	0.97	1.00	2167	4318	482	109	7	6	220	330
π_6	0.93	1.00	2154	3612	454	72	41	8	200	300
π_7	0.95	0.94	2140	4318	467	185	44	7	220	320
π_8	0.98	1.00	2408	4361	475	340	41	4	205	325
π_9	0.97	1.00	2400	3499	513	212	1	8	230	295
π_{10}	0.98	1.00	2277	4322	481	230	28	5	230	360

C. COMPARISON WITH EXISTING DECISION-SUPPORT PRACTICES

In this section, we compare our approach with existing DSS practices. First, we recall that the main contribution of our paper is a novel architecture that addresses a key limitation of current optimization-based DSS: their lack of flexibility in accommodating new requirements, preferences, or constraints that may emerge during the system's lifecycle. A natural question is: *how does our approach compare with current DSS practices?* The answer depends on the specific decision-making context, as such settings vary widely. However, broadly speaking, when new requirements arise, decision makers typically have three main options:

1) dismiss the DSS *or*
2) commission a software vendor to overhaul the DSS, which is often costly and time-consuming *or* (which is the most common case)
3) attempt to adjust weights or parameters in the hope of steering the system toward solutions that better align with their updated preferences.

To illustrate the challenge with the third option, consider the following example: a DSS originally designed for vehicle routing with the objective of minimizing total travel distance. At some point, a new priority emerges to better balance the workload among drivers. To incorporate this, the decision maker might run the existing DSS and obtain, for instance, three routes with durations of 7h50', 6h10', and 4h25'.

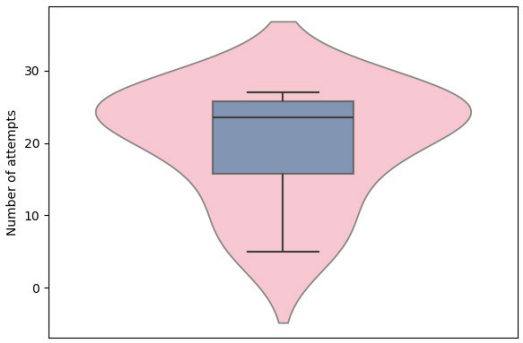

FIGURE 2. Boxplot and violin plot showing how frequently decision-makers attempted to adjust DSS parameters in practice, based on the experiment described in Section IV-C.

Seeking greater balance, they may attempt to reduce the allowable shift duration for the first driver (e.g., from 8 hours to 7 hours) and rerun the DSS. However, this might yield a new plan that remains unbalanced - say, with route durations of 6h45', 7h50', and 4h25' - or even produce an infeasible solution.

As a result, the decision maker may need to iteratively test various parameter configurations (in this case, shift durations) to find one that better satisfies the new balancing requirement while still keeping travel distances low. This process typically relies heavily on the decision maker's domain expertise, is time-consuming, and offers no guarantee of finding a satisfactory solution - even when one exists.

With that in mind, it is clear that the benefits offered by our approach are difficult to quantify, as they depend heavily on the specific decision-making context, as well as on the experience and mindset of the decision maker. That said, we conducted the following experiment to provide an illustrative comparison.

We considered a basic version of the VRP, for which a traditional DSS was available, and introduced a set of new requirements, preferences, and constraints, all expressed in natural language. Ten individual not involved in this research were asked to act as decision-makers and tasked with adjusting the VRP algorithm parameters - such as driver shift durations, customer time windows, and customer demands - in an attempt to satisfy the new requirements.

The results showed that, on average, 19.8 attempts were needed to arrive at a satisfactory solution. However, there was significant variability: a few individuals were able to identify promising parameter combinations quickly (either due to skill or luck), while the majority required many more iterations (see the boxplot and violin plot in Figure 2).

Naturally, we do not claim that these figures are generalizable. The purpose of this experiment was simply to illustrate the challenges faced under current DSS practices and to highlight, through contrast, the potential advantages of our more flexible approach.

V. CONCLUSION

This paper introduced a novel framework designed to address the rigidity commonly found in traditional optimization-based DSS. By integrating recent advancements in large language models, the proposed hybrid approach enables natural language interaction, adaptive reconfiguration of algorithmic components, and preference-based decision support, all while offering transparent, human-understandable explanations for system behavior.

A key strength of the framework lies in its built-in fallback mechanisms, which require human validation before deployment - ensuring both safety and accountability in critical applications. Computational experiments conducted on a ''rich'' Vehicle Routing Problem demonstrated that the system maintains high standards of flexibility, robustness, and explainability without compromising on computational efficiency or solution quality.

Importantly, the inclusion of LLMs introduces only minimal overhead, suggesting that such hybrid systems can scale well in real-world decision-making environments.

While the experimental evaluation focused on a complex variant of the Vehicle Routing Problem, the architecture and underlying principles of the framework are general and modular, making them applicable to a wide range of problem domains. In particular, domains involving combinatorial decision-making, dynamic constraints, and evolving user preferences - such as scheduling, logistics, resource allocation, or strategic planning - may benefit from this approach. A broader empirical validation across diverse application areas remains a valuable direction for future research.

Future work should primarily address the main limitation of our current methodology: as the number of alternative plans m grows exponentially - common in many metaheuristic frameworks - a single-step, one-shot MCDM comparison becomes impractical. An iterative approach, where each newly generated plan is evaluated against a limited set of non-dominated alternatives, may offer a more scalable solution by keeping computation time and memory usage within reasonable bounds. Developing and validating such a procedure remains an open direction for future research.

Additionally, future work should investigate more autonomous, agentic configurations of the framework, with the goal of enabling proactive, context-aware decision support suitable for a broader range of complex operational environments.

REFERENCES

[1] J. P. Shim, M. Warkentin, J. F. Courtney, D. J. Power, R. Sharda, and C. Carlsson, ''Past, present, and future of decision support technology,'' *Decis. Support Syst.*, vol. 33, no. 2, pp. 111–126, Jun. 2002.

[2] R. H. Bonczek, C. W. Holsapple, and A. B. Whinston, *Foundations of Decision Support Systems*. New York, NY, USA: Academic, 2014.

[3] R. L. Ackoff, ''The future of operational research is past,'' *J. Oper. Res. Soc.*, vol. 30, no. 2, pp. 93–104, Feb. 1979.

[4] D. Arnott and G. Dodson, ''Decision support systems failure,'' in *Handbook on Decision Support Systems 1*, 2008, pp. 763–790.

[5] P. Toth and D. Vigo, *The Vehicle Routing Problem*. Philadelphia, PA, USA: SIAM, 2002.

[6] A. Mor and M. G. Speranza, ''Vehicle routing problems over time: A survey,'' *4OR*, vol. 18, no. 2, pp. 129–149, Jun. 2020.

[7] P. Kumar, ''Large language models (LLMs): Survey, technical frameworks, and future challenges,'' *Artif. Intell. Rev.*, vol. 57, no. 10, p. 260, Aug. 2024.

[8] S. Schulhoff et al., ''The prompt report: A systematic survey of prompt engineering techniques,'' 2024, *arXiv:2406.06608*.

[9] L. Berti, F. Giorgi, and G. Kasneci, ''Emergent abilities in large language models: A survey,'' 2025, *arXiv:2503.05788*.

[10] D. G. Bobrow, ''A question-answering system for high school algebra word problems,'' in *Proc. AFIPS*, Oct. 1964, pp. 591–614.

[11] S. S. Sundaram, S. Gurajada, D. Padmanabhan, S. S. Abraham, and M. Fisichella, ''Does a language model 'understand' high school math? A survey of deep learning based word problem solvers,'' *WIREs Data Mining Knowl. Discovery*, vol. 14, no. 4, p. 1534, Jul. 2024.

[12] E. Davis, ''Mathematics, word problems, common sense, and artificial intelligence,'' *Bull. Amer. Math. Soc.*, vol. 61, no. 2, pp. 287–303, Apr. 2024.

[13] A. Kamath and R. Das, ''A survey on semantic parsing,'' 2018, *arXiv:1812.00978*.

[14] P. Jiang and X. Cai, ''A survey of semantic parsing techniques,'' *Symmetry*, vol. 16, no. 9, p. 1201, Sep. 2024.

[15] R. Ramamonjison, T. Yu, R. Li, H. Li, G. Carenini, B. Ghaddar, S. He, M. Mostajabdaveh, A. B.-Dehkordi, Z. Zhou, and Y. Zhang, ''Nl4opt competition: Formulating optimization problems based on their natural language descriptions,'' in *Proc. NeurIPS*, 2023, pp. 189–203.

[16] R. Ramamonjison, T. T. 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 *Proc. NeurIPS Competition Track*, 2023, pp. 189–203.

[17] OpenAI. (2024). *GPT-4*. Accessed: May 29, 2025. [Online]. Available: https://www.openai.com/research/gpt-4

[18] T. Ahmed and S. Choudhury, ''LM4OPT: Unveiling the potential of large language models in formulating mathematical optimization problems,'' *Inf. Syst. Oper. Res.*, vol. 62, no. 4, pp. 559–572, Nov. 2024.

[19] N. Astorga, T. Liu, Y. Xiao, and M. van der Schaar, ''Autoformulation of mathematical optimization models using LLMs,'' 2024, *arXiv:2411.01679*.

[20] M. Mostajabdaveh, T. T. Yu, R. Ramamonjison, G. Carenini, Z. Zhou, and Y. Zhang, ''Optimization modeling and verification from problem specifications using a multi-agent multi-stage LLM framework,'' *INFOR: Inf. Syst. Oper. Res.*, vol. 62, no. 4, pp. 599–617, Nov. 2024.

[21] S. Wasserkrug, L. Boussioux, D. den Hertog, F. Mirzazadeh, I. Birbil, J. Kurtz, and D. Maragno, ''From large language models and optimization to decision optimization CoPilot: A research manifesto,'' 2024, *arXiv:2402.16269*.

[22] R. Thind, Y. Sun, L. Liang, and H. Yang, ''OptimAI: Optimization from natural language using LLM-powered AI agents,'' 2025, *arXiv:2504.16918*.

[23] A. Talebi, ''Large language model-based automatic formulation for stochastic optimization models,'' Dept. Integr. Syst. Eng., Ohio State Univ., Columbus, OH, USA, Tech. Rep., 2025. [Online]. Available: https://www.researchgate.net/publication/393521260_Large_Language_Model-Based_Automatic_Formulation_for_Stochastic_Optimization_Models

[24] Z. Huang, G. Shi, and G. S. Sukhatme, ''Can large language models solve robot routing?'' 2024, *arXiv:2403.10795*.

[25] R. Ramírez-Rueda, E. Benítez-Guerrero, C. Mezura-Godoy, and E. Bárcenas, ''A systematic literature review of 10 years of research on program synthesis and natural language processing,'' *Program. Comput. Softw.*, vol. 50, no. 8, pp. 725–741, Dec. 2024.

[26] D. Palla and A. Slaby, ''Evaluation of generative AI models in Python code generation: A comparative study,'' *IEEE Access*, vol. 13, pp. 65334–65347, 2025.

[27] M. Gendreau and J.-Y. Potvin, *Handbook of Metaheuristics*, vol. 2. Cham, Switzerland: Springer, 2010.

[28] E. K. Burke, M. Gendreau, M. Hyde, G. Kendall, G. Ochoa, E. Özcan, and R. Qu, ''Hyper-heuristics: A survey of the state of the art,'' *J. Oper. Res. Soc.*, vol. 64, no. 12, pp. 1695–1724, Dec. 2013.

[29] X. Wu, D. Wang, C. Wu, L. Wen, C. Miao, Y. Xiao, and Y. Zhou, ''Efficient heuristics generation for solving combinatorial optimization problems using large language models,'' 2025, *arXiv:2505.12627*.

[30] D. E. Goldberg and W. Shakespeare, ''Genetic algorithms,'' in *Handbook Metaheuristics*. Boston, MA, USA: Springer, 2002, pp. 109–139.

[31] K. Li, F. Liu, Z. Wang, X. Tong, X. Han, M. Yuan, and Q. Zhang, "ARS: Automatic routing solver with large language models," 2025, *arXiv:2502.15359*.
[32] S. T. Windras Mara, R. Norcahyo, P. Jodiawan, L. Lusiantoro, and A. P. Rifai, "A survey of adaptive large neighborhood search algorithms and applications," *Comput. Operations Res.*, vol. 146, Oct. 2022, Art. no. 105903.
[33] F. Da Ros, M. Soprano, L. Di Gaspero, and K. Roitero, "Large language models for combinatorial optimization: A systematic review," 2025, *arXiv:2507.03637*.
[34] G. D. Konstantakopoulos, S. P. Gayialis, and E. P. Kechagias, "Vehicle routing problem and related algorithms for logistics distribution: A literature review and classification," *Oper. Res.*, vol. 22, no. 3, pp. 2033–2062, Jul. 2022.
[35] J. Zhang and T. V. Woensel, "Dynamic vehicle routing with random requests: A literature review," *Int. J. Prod. Econ.*, vol. 256, Feb. 2023, Art. no. 108751.
[36] E. Angelelli, N. Bianchessi, R. Mansini, and M. G. Speranza, "Management policies in a dynamic multi period routing problem," *Innovations in distribution logistics*. Berlin, Germany: Springer, 2009, pp. 1–15.
[37] M. Wen, J.-F. Cordeau, G. Laporte, and J. Larsen, "The dynamic multi-period vehicle routing problem," *Comput. Operations Res.*, vol. 37, no. 9, pp. 1615–1623, Sep. 2010.
[38] M. Albareda-Sambola, E. Fernández, and G. Laporte, "The dynamic multiperiod vehicle routing problem with probabilistic information," *Comput. Operations Res.*, vol. 48, pp. 31–39, Aug. 2014.
[39] M. W. Ulmer, N. Soeffker, and D. C. Mattfeld, "Value function approximation for dynamic multi-period vehicle routing," *Eur. J. Oper. Res.*, vol. 269, no. 3, pp. 883–899, Sep. 2018.
[40] (2024). *LangChain: Build, Run, and Manage LLM Applications*. Accessed: May 29, 2025. [Online]. Available: https://www.langchain.com

GIANPAOLO GHIANI received the M.S. degree in electronics engineering and the Ph.D. degree in computer engineering from the University Federico II, Naples, Italy. He was a Postdoctoral Researcher at the Groupe d'Études et de Recherche en Analyse des Décisions (GERAD), Université de Montréal, Montreal, Canada, from 1998 to 1999. He is currently a Full Professor of operations research at the University of Salento, Lecce, Italy. His main research interests include decision support systems, such as methodologies at the boundary between operations research and generative artificial intelligence, with an emphasis on applications to logistics systems planning. On these themes, he has published over 80 articles in a variety of international journals. He has also co-authored the textbook *Introduction to Logistics Systems Planning* (Wiley, 2022). His doctoral thesis was awarded the Transportation Science Dissertation Award from the Institute for Operations Research and the Management Science (INFORMS), in 1998.

EMANUELE MANNI received the M.S. degree in computer engineering from the University of Lecce, Lecce, Italy, in 2004, and the Ph.D. degree in operations research from the University of Calabria, Italy, in 2008. He is currently an Associate Professor of operations research at the University of Salento. He is the author of more than 30 articles. His research focuses on methods combining operations research and artificial intelligence, with applications in logistics systems planning and control. He is exploring how large language models can be integrated with traditional optimization algorithms. He is a member of Italian Association of Operations Research as well as the Institute for Operations Research and the Management Science (INFORMS).

SANDRO ZACCHINO received the Dr.Eng. degree in computer engineering from the University of Lecce (now the University of Salento), Italy, and the Ph.D. degree in operations research from the University of Calabria, Rende, Italy. He was a Postdoctoral Researcher at the University of Salento, where he contributed to several projects on computational geometry applied to leather nesting problems, optimal workforce assignment, and machine tooling and scheduling. From 2015 to 2016, he was a Technical Specialist at Altair Engineering. Since 2017, he has been a Computer Scientist at the University of Salento. His current research interest includes the integration of generative models with decision support systems.