City-LEO: Toward Transparent City Management Using LLM with End-to-End Optimization

Zihao Jiao
School of Computer Science and Artificial Intelligence
Beijing Technology and Business University
Beijing, China

Mengyi Sha
Department of Industrial Engineering
Tsinghua University
Beijing, China

Haoyu Zhang
School of Computer Science and Artificial Intelligence
Beijing Technology and Business University
Beijing, China

Xinyu Jiang
Department of Industrial Engineering
Tsinghua University
Beijing, China

Wei Qi
Department of Industrial Engineering
Tsinghua University
Beijing, China
qiwei.0216@gmail.com

Abstract

Existing operations research (OR) models and tools play indispensable roles in smart-city operations. However, the adoption of conventional OR tools confronts some challenges: On one hand, the highly intricate nature of urban operations introduces difficulties in accurately modeling the problems in city management. On the other hand, the inherent intransparency of conventional OR models, coupled with the requirement for specialized OR expertise, also sets barriers in the development and implementation processes. Amidst these challenges, Large Language Model (LLM) is rapidly evolving and offers a promising path to bridge the gap between OR theory and practical implementation. Nevertheless, existing LLM-based OR techniques exhibit several limitations in terms of relevance and accuracy. To alleviate the limitations, in this paper we make an early attempt to propose an innovative LLM-embedded agent, City-LEO. The agent aims to conduct human-like decision process and deliver more relevant and accurate solutions to tackle city management problems. Specifically, to enhance computational tractability and relevance to users' queries, based on a pre-specified full-scale optimization model, City-LEO leverages LLM's logical reasoning capability on prior knowledge to effectively scope down the users' requirements. In addition, to promote transparency and account for environmental features, City-LEO incorporates an End-to-End (E2E) framework to synergize prediction and optimization process. The feature-to-decision framework is conducive to coping with environmental uncertainties and involving social concerns. In the case study, we employ City-LEO in the operations management of an electric-bike sharing (EBS) system. We design relevance tests and accuracy tests to evaluate the performance of City-LEO, and the numerical results demonstrate that City-LEO has superior performance when benchmarked against the full-scale optimization problem. With less computational time, City-LEO generates more relevant solutions to the users' requirements without significantly compromising accuracy. In a broader sense, City-LEO showcases the potential of developing LLM-embedded OR tools for smart-city operations management.

Keywords LLM-based agent · end-to-end optimization · smart-city operations · electric-bike sharing

1 Introduction

Operations research (OR) models and tools, such as commercial solvers including Gurobi and Cplex, are being used extensively in smart-city transportation management to prescribe planning and operational decisions (Qi and Shen 2019, Zhang et al. 2022a, Gayialis et al. 2022, Chen and Xiong 2023). For urban bus transit, bike-sharing and ride-hailing systems, OR models and tools facilitate the decision-making processes and generate planning and scheduling strategies to promote operational efficiency. Nevertheless, as (Bettencourt 2024) in *Science* points out, the urban systems encompass a multitude of interrelated or uncertain factors, and are usually highly complex to model accurately. The highly intricate nature of urban operations, coupled with the heightened need for specialized OR applications, results in a significant disparity between theory and practical implementation. The adoption of OR tools also confronts challenges that stem from inherent *intransparency* in development and implementation processes. Generally, city operators possess a deep understanding of urban operations mechanisms but lack a fundamental understanding of OR expertise. Conversely, the developers of OR tools have a solid grasp of OR methodologies, but often lack knowledge of the practical operational principles of urban systems. Moreover, conventional OR tools cause information asymmetry between the city operators and the residents. The city operators hold a dominant position in the decision-making process and control the OR tools, while the process appears to be a black box for the residents.

Given these challenges at hand, large language models (LLMs) hold promise to empower city operators to wield OR models and solvers toward enabling more transparent city management. The boom of LLMs, such as GPT and Ollama, marks a breakthrough in artificial intelligence. Based on deep learning techniques, LLMs offer a path to meaningful utilization of the growing amounts of municipal data and understanding, summarizing, generating, and predicting new content. LLMs have recently penetrated into an array of application fields. Several vertical fields of LLMs, such as the legal LLM ChatLaw and the financial LLM XuanYuan 2.0 (Zhang and Yang 2023), have emerged to prescribe intelligent decisions in their respective fields. As a promising technique for assisting humans in multiple tasks, LLMs have also been tried in OR applications. In practice, there have been two main pathways for implementing LLMs:

- *LLMs as optimizers*. This technique leverages the in-context learning LLM (Dong et al. 2022) to directly address an optimization problem conversationally without invoking an optimization model. By integrating chain of thought (CoT) prompts (Kojima et al. 2022), the LLM's reasoning abilities can be harnessed to directly tackle and resolve an optimization problem. LLMs as optimizers do not require extensive knowledge of specific details or specialized OR expertise. The technique only needs a few prompts (e.g., *Let's think step by step* (Wei et al. 2022)) to provide solutions for user-defined problems.
- *LLM agents*. In order to enhance the transparency and accessibility of the decision-making process, another straightforward strategy would be to develop an LLM agent that is capable of generating optimization models (in the form of callable APIs). Some recent preliminary works propose to integrate OR optimization techniques with LLM into an *Agent*, and these LLM-based agents mainly focus on deterministic mixed integer programming (MIP) models (Kasneci et al. 2023, Li et al. 2023a, AhmadiTeshnizi et al. 2023).

Nevertheless, existing LLM-based OR techniques exhibit limitations in terms of *relevance* and *accuracy*. The extent to which we can extrapolate from these LLM capabilities to the intricate realm of city management remains unclear. Although *LLMs as optimizers* negate the need for OR expertise, guaranteeing the *accuracy* of solutions without the aid of rigorous modeling and solvers is challenging. For example, the experiments of LLM optimizers show that there is still an 11% gap between the achieved solution and the optimal solution in a 50-node TSP problem (Yang et al. 2023). On the other hand, *LLM agents* are generally restricted to dealing with simple and well-defined problems. However, city operators usually confront highly intricate issues, and may struggle to accurately match suitable models and code with vague and macro-level queries, which may result in *irrelevant* optimization model recommendations. Consequently, agents associated with callable MIPs (e.g., (AhmadiTeshnizi et al. 2023) and (Li et al. 2023a)) may lack the flexibility to generate adequate functionalities and accommodate city operators' diverse requests. Moreover, agents with callable MIP often overlook environmental uncertainties, underutilize historical data and involve limited parameters and decisions, which hinders their applicability and effectiveness in addressing complex urban operational problems. On the whole, as Figure 1 illustrates, there is a dilemma between the relevance and accuracy of LLM-based OR tools in city management. The dilemma lies in the difficulty of simultaneously achieving complex problem-solving and optimality. Specifically, in most cases, it is difficult to prescribe a model in OR applications that can match users' diverse queries based on large-scale complex parameters and decision

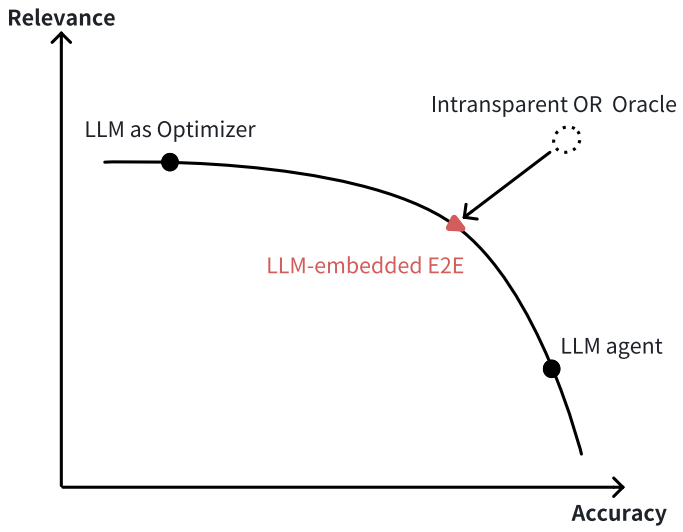


Figure 1: Dilemma between Relevance and Accuracy of LLM-based OR Tools in City Management

types and then use a solver to get efficient and exact solutions. The limitations of *LLMs as optimizers* and *LLMs agents* urge us to identify a sweet middle ground to get close to the intransparent (and often intricate) OR Oracle (see, Figure 1) without much compromising the relevance or accuracy. Here, we are motivated to propose City-LEO, an **L**LM-based agent joint with **E**2E **O**ptimization, capable of providing reasonably relevant and accurate operational strategies to city operators through transparent interactions:

- To ensure accuracy and relevance of the LLM-embedded optimization tool, we develop a full-scale optimization model first. The pre-defined model involves more constraints and covariates (e.g. features and decision variables), and allows for a nuanced characterization of the complex operations of a specific urban system. In this way, the comprehensive model incorporates domain knowledge and serves as the bedrock of our proposed framework.
- To further enhance relevance and computational tractability, we propose a human-like decision-making process. The process allows City-LEO to leverage LLM's reasoning capability based on prior knowledge, so as to effectively *scope down* the pre-specified full-scale optimization model based on users' requirements. Such problem-scoping operations allow for focusing on a careful choice of appropriate parameters and decisions from a diverse set of potential candidates for problem modeling to minimize irrelevance and computational expenditure. Additionally, the daily operational goals proposed by city operators usually focus on *neighborhood-level* problems, which typically do not necessitate redundant full-scale optimization models as seen in traditional OR applications. Thus, City-LEO is conducive to effectively scoping down the size of the large-scale optimization problem to accommodate users' diverse requirements, while lowering the expertise barrier for city operators in urban governance.
- In order to further enhance transparency and account for environmental uncertainties and contextual features, we extend from the literature an E2E framework that outputs decisions directly from input covariates. Specially, the E2E framework integrates prediction with optimization processes, and conducts MIP optimization over a Random Forest (RF) objective function with general polyhedral constraints. On the one hand, the tool entails adopting a large number of features into the optimization model, which has the potential to increase the degree of relevance to the user's requirements and then mitigate decision-making bias. On the other hand, random forest is powerful to depict the complex and unknown nonlinear relationships between features and macro-level indicators, and the hierarchical tree structure can reflect the decision paths in random forest and makes the E2E framework more interpretable and transparent.

Taken together, we summarize the contributions of our work as follows:

(1) We make an early attempt to propose an innovative LLM-embedded agent, City-LEO, with the aim of delivering more *relevant* and *accurate* solutions to tackle city management problems. The agent also enhances the accessibility of OR tools through conversational interactions, and promotes the transparency and efficiency of the decision-making processes.

(2) We expand the nascent agent framework of synergizing LLMs and OR in a methodology-wise approach. In particular, (i) City-LEO is conducive to effectively scoping down the pre-defined large-scale optimization problem to accommodate users' diverse requirements. The agent leverages the LLM's logical reasoning ability based on prior knowledge to redefine the problem and conduct human-like decision-making processes. The scope-down policy not only enhances the computational tractability of the optimization model, but also elevates the relevance of the solutions in addressing city management problems. (ii) We integrate an E2E modeling framework with a pre-trained random forest to cope with city management problems in an uncertain operational environment. Different from existing MIP agents, the E2E framework integrates estimation with optimization, and offers an opportunity to generate decisions directly from feature data, which is more familiar to residents. The feature-to-decision framework promotes data utilization (Qi et al. 2023), and also leads to more transparent and responsible decision processes by incorporating indicators to reflect social concerns.

(3) We numerically test City-LEO in the operations management of an urban electric bike sharing (EBS) system. Specifically, the agent focuses on optimizing the e-bike rebalancing and battery swapping operations to mitigate the imbalance between supply and customers' demands. Our case study demonstrates that the City-LEO has superior performance in terms of accuracy and relevance. By benchmarking against the full-scale MIP model, the numerical results validate the effectiveness of City-LEO in terms of computational time and the quality of the solutions. Moreover, the results demonstrate that our proposed approach holds promise of using LLM to amplify the value of OR models in addressing complex smart-city operations problems.

The remainder of this paper is organized as follows. Section 2 reviews the related literature. Section 3 provides a formal introduction to the structure of the LLM agent. Section 4 presents the E2E models for the EBS system management. Section 5 discusses numerical results. Section 6 concludes the paper. Notation tables, methodology review, additional experimental settings and sample of prompts are available in the online appendices. We also publish our data and code source related to the E2E model, City-LEO agent, and necessary materials files to Hugging-face (please see City-LEO) for the convenience of readers.

2 Literature Review

Synergizing OR and LLM on tackling complex urban management problems is an important and intriguing paradigm for future smart cities, but so far has not been examined in the literature to our knowledge. Existing literature has studied both utilizing the reasoning ability of LLM to solve optimization problems heuristically and integrating LLM with professional tools by constructing sophisticated structures for agents, yet with little attention on LLM-embedded optimization. We review the related literature in multiple academic fields, and then highlight the distinction of our proposed methodology. To familiarize the OR audience with a more general background on using LLMs for optimization, we provide an extended literature survey on the recent development of In-context learning in Section 2.3.

2.1 LLM Agents

LLM agents (Wu et al. 2023, Zhao et al. 2024), which integrate externally callable domain knowledge and flexible functions, are capable of mitigating the low accuracy and token-limitation (as demonstrated in Section 2.3) issues associated with in-context learning. Typically, LLM agents utilize external functions to generate various extensions, which can be classified as *LLM agents with function calling* and *Autonomous language agents* (ALA, see Section 2.4 for details).

LLM Agents with Function Calling. The first approach entails directly utilizing pre-packaged external functions, referred to as "function calling" (LLM-FC). To enhance the accuracy of the responses, LLM-FC leverages the reasoning capabilities of LLM to match the user's query with domain knowledge (e.g., open data source, commercial solvers like Gurobi and Cplex), which is pre-defined as structural functions in an external API (Ge et al. 2024). In the context of optimization, the main concept underlying LLM-FC is not to substitute optimization technology with LLMs but rather to employ optimization solvers in tandem with LLMs. The agent utilizes a specific and specialized template to identify users' queries and generate exact solutions to offer responses. For instance, through commercial solver agents (AhmadiTeshnizi et al.

2023), users can adopt natural language to invoke code generation commands in the solver. (Li et al. 2023a) proposes agents and a response architecture to cope with "what if" questions. The agent modifies the input to the optimization solver in a suitable manner and subsequently re-executes the solver to generate a solution.

Acting as a "translator" between users and expertise OR tools, the paradigm associated with LLM agents confronts with professional barriers yet: Typically, city administrators who lack expertise in optimization theory struggle to accurately match suitable models and precise code with vague and macro-level queries, resulting in ineffective optimization recommendations. For example, prior to proceeding, LLM-FC requires the user to provide an exceedingly accurate and detailed depiction of the issue, including quantifiable objectives, key parameters (e.g. unit cost and resources) and constraints to represent the problem setting (AhmadiTeshnizi et al. 2023, Li et al. 2023b).

However, in practice, it is challenging for users to provide precise and detailed description of the problem. Thus, the issue sets barriers to extracting useful information from users' queries and constructing "accurate" and "feasible" models for LLM-FC, which may result in inefficient responses and weakening the model's potential for practical use. For instance, (AhmadiTeshnizi et al. 2023) shows that the success rate (i.e., the ratio of outputs that satisfies constraints and can find the optimal solution) is less than 80% based on GPT-4.0 function calling. Hence, the subsequent limitations of employing such agents cause possible discrepancies between users' desired responses and the generated code. Consequently, MIP-embedded agents possibly prescribe wrong and opaque (if not infeasible) decisions for citywide management. We summarize the limitations of the existing approaches as follows. Area-specific MIP agents (e.g. LLM-FC and ALA), relying on deterministic assumptions, are unable to provide a diverse range of solutions that effectively meet the users' requirements. The efficacy of the agents are limited by the pre-defined parameters and decision variables. Hence, it is imperative to develop decision-support LLM agents that can effectively address users' queries and comprehensively consider the attributes of urban systems and environments.

2.2 End-to-End Optimization

LLM agents have some limitations in dealing with various practical problems in uncertain environments. (i) If a user's query does not cover some key parameters or conditions of the MIP model, the code generation is either incomplete or ambiguous. (ii) Users' open-ended queries consistently cause variable problem settings, which poses challenges to deterministic models. Moreover, deterministic MIP optimization models incur biased solutions due to inefficient estimation of uncertainties. In contrast, E2E optimization offers a promising way for LLM-assisted optimization to generate relevant and accurate solutions. E2E optimization refers to outputting decisions directly from the input covariates. The feature-to-decision mapping integrates prediction and optimization process.

We identify two branches of research on feature-to-decision mapping: (i) One-step mapping is an integrated version of learning and optimization to search for the best-performing decisions. (Qi et al. 2023) involves deep learning models in the E2E framework to generate the multi-period inventory replenishment strategies directly from the input features. (ii) Two-step mapping means sequential learning and optimization process, which initially utilizes given covariates and trained model to estimate the conditional distribution of the uncertain parameters, and then tackles an associated optimization problem to derive the optimal solution (Elmachtoub and Grigas 2022). For instance, (Biggs et al. 2022) propose to reformulate random forests along with operational polyhedral constraints into 0-1 mixed integer programs. The prediction in this approach ignores the properties of subsequent optimization models, which may result in error propagation between estimation and optimization and suboptimal solutions (Qi and Shen 2022).

Therefore, in our proposed LLM-based optimization tool, we adopt E2E optimization to address open-ended queries about EBS management for the following reasons: First, extending the approach of (Biggs et al. 2022), utilizing pre-trained random forests can avoid repetitive training in dealing with diverse user queries. Second, the objectives can include indicators (e.g., traffic congestion metrics) related to social concerns and contribute to mitigating negative social externalities. For city administrators, the tool based on E2E optimization is user-friendly and does not require extensive specialized knowledge.

2.3 In-context Learning

One topic that aligns with our research is In-context learning. Research in this area is mostly aimed at promoting LLMs' abilities to solve area-specific problems (such as mixed integer programming in our context) through *Prompt Engineering*, represented as in-context learning methods (Dong et al. 2022), which include the zero-shot chain of thought (CoT) (Kojima et al. 2022), few-shot CoT (Wei et al. 2022), and Auto-CoT

(Zhang et al. 2022b). Specifically, zero-shot CoT refers to the capacity of LLMs to perform logical reasoning without any prior training examples, whereas few-shot CoT entails offering the model with a limited number of examples to direct its reasoning. The auto-CoT samples questions and develops chains of reasoning to construct demonstrations. One of the relevant research (Yang et al. 2023) leverages the capabilities of LLMs as optimizers, designs a meta-prompt, and solves optimization problems interpreted from natural language. Nevertheless, the results show a poor level of accuracy in solving the optimization problem. As an illustration, the LLM optimizer experiment still shows an 11% optimality gap from the best solution in a 50-node TSP problem, as determined using oracle solutions.

The low accuracy observed in the aforementioned CoT methods is caused by the model's unstable reasoning results, resulting in hallucinatory phenomena such as factual inaccuracies and logical errors. (Chu et al. 2023) attributes the issue to the absence of high-quality artificial annotations and limited reasoning paths. To mitigate hallucinatories, (Yao et al. 2023a) proposed a novel framework called the tree of thought (ToT), which enables the LLM to take into account various reasoning paths. ToT greatly enhances the interpretability and reasoning capabilities of the reasoning process by adding more prompts. However, limited by capacity and resources, LLMs do not have enough tokens to deal with longer prompts. TOT is subject to significantly more constraints when choosing activities and generating tailored text recommendations for each query.

2.4 Autonomous Language Agent

ALA has the ability to automatically handle objective-oriented multi-step tasks (Park et al. 2023), and enables autonomous problem-solving, allowing it to handle specific objectives and tasks independently rather than simply responding to users' queries.

In this vein, (Yao et al. 2023b) utilizes verbal "feedback", specifically self-reflection, to assist agents in acquiring knowledge from previous instances of failure. Furthermore, the feedback can be defined as gradient reduction of optimization objectives, thereby contributing to solving MIP model. For instance, (Yao et al. 2022) propose a reinforced large language agent by adopting environment-specific rewards. The agent utilizes the rewards from various contexts and tasks to fine-tune a pre-trained LLM. The LLM is then used to improve the language agent prompt by summarizing the main reason for the previous unsuccessful attempts and suggest action plans. Nevertheless, the ALA continues to exhibit low accuracy and significant token expenses due to the presence of a large-scale ICL prompts.

3 The City-LEO Agent

This section illustrates the task flow framework and an EBS case for our proposed City-LEO agent. Through the ① *Problem matcher*, the agent matches (i.e., first-stage scoping down) users' natural language queries Q (See Appendix A for all notations) with appropriate area-specific agents (i.e., callable tool APIs). It consists of the following three critical components: ② The *Query-relevant objective generator* is the process of generating an Query-relevant objective (QR-obj) function $f(\mathbf{x};\mathbf{w}) \overset{\text{def}}{=} \mathbf{LLM}(Q; P^{\text{IG}}) : \mathbb{R}^{|\mathcal{X}|\times|\mathcal{W}|} \to \mathbb{R}_+$ for input Q by incorporating pre-defined prompts P^{IG}, where $\mathcal{X}, \mathcal{W}$ denote the feasible domain of decision variables $\mathbf{x}$ and set of parameters $\mathbf{w}$, respectively. Function $f(\mathbf{x};\mathbf{w})$ serves as a secondary objective in E2E optimization problem $\min_{\mathbf{x}\in\mathcal{X}} g(\mathbf{x};\mathbf{w})$. ③ *Problem tailor* is adopted for second-stage scoping down the feasible domain $\mathcal{X}$ of decision variables $\mathbf{x}$, particularly searching for *parameterized variables* $\mathbf{x}' \in \mathcal{X}$ in the E2E optimization programming $\min_{\mathbf{x}\in\mathcal{X}} g(\mathbf{x};\mathbf{w})$. Parameterized variables refer to the decision variables that are less relevant to the user's query, based on an inference of historical information. And we assign historical values to parameterized variables in the form of equation constraints. ④ The *E2E model* comprises the pre-trained random forest model, along with the associated MIP model. The agent incorporates a code safeguard that performs iterative examinations based on the optimization language rule of Gurobi. Figure 2 provides a comprehensive illustration of the agent framework:

Step 1: Problem Matcher. The user (e.g., the city operators, the EBS system managers, etc.) submits a natural language query Q based on a certain target in a specific area (e.g., *How to decrease the proportion of shared e-bikes with low SOC in parking spots 5, 6, and 7?*). Such a query is subsequently processed by the *problem matching agent* (see Figure 2 ①) to determine suitable area-specific agents (e.g., EBS, bus line analysis, or other city management area agents). Such area-specific agents encompass QR-obj generators, problem tailors, the E2E models, and response-prompters that are specifically associated with the given problem.

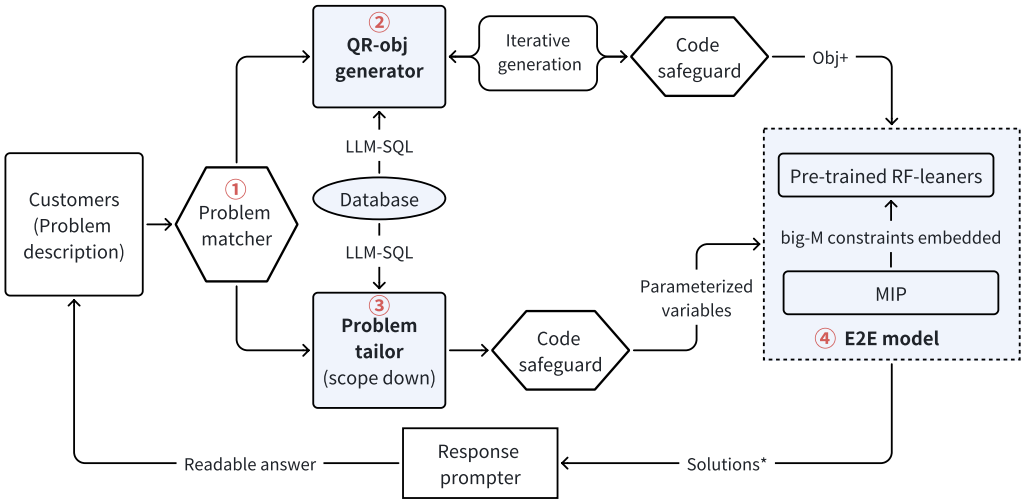


Figure 2: Framework of the City-LEO Agent

Step 2: QR-obj Generator. The user query is subsequently processed by the QR-obj generator within area-specific agents that have been matched. The QR-obj generator (see Figure 2 ②) formulates natural language queries Q into a convex QR-obj function $f(\mathbf{x}; \mathbf{w})$ of decisions $\mathbf{x}$ (e.g., number of dispatched e-bikes or number of swapped depleted batteries) and parameters $\mathbf{w}$ in the E2E programming $\min_{\mathbf{x}\in\mathcal{X}} g(\mathbf{x}; \mathbf{w})$. The QR-obj $f(\mathbf{x}; \mathbf{w})$ is then incorporated into the objective of MIP for re-optimization. In addition, the retrievable historical operational databases $\mathcal{D}$ (in the form of a callable API with LLM-generated structured query language (SQL)) provide adequate operational information for QR-obj generator. It further strengthen the relevancy and significance of the produced QR-obj to users' query. Following that, a *Code safeguard* will iteratively examine the output code of $f(\mathbf{x}; \mathbf{w})$ to ensure that it complies with the code rules of solvers like Gurobi, CPLEX, etc.

Step 3: Problem Tailor. The main task of the problem tailor is to scope down the macro-level queries Q into specific subsets of decisions $\hat{\mathbf{x}} \in \mathcal{X}\backslash\mathbf{x}'$ and relevant factors rather than considering the entire collection of decisions and constraints in the E2E model $\min_{\mathbf{x}\in\mathcal{X}} g(\mathbf{x}; \mathbf{w})$. The conversion is completed by using prompted $\mathbf{LLM}(Q; \mathcal{D}, P^{\text{PP}})$ based on empirical record analysis $\mathcal{D}$ and prompt P^{PP}. Specifically, as seen in Algorithm 1, the problem tailor first leverages LLM-generated SQL instructions to obtain compatible query-related information in databases $\mathcal{D}$, particularly those containing large-scale empirical operational data $(\bar{\mathbf{x}}, \bar{\mathbf{w}})$. Furthermore, the problem tailor examines which variables are associated with the user's query based on historical data and subsequently generates the most relevant decision variables $\hat{\mathbf{x}}$. This process can be formulated into an optimization problem to obtain the most relevant decision subset $\hat{\mathbf{x}}$ about Q:

$$\hat{\mathbf{x}}_t = \arg \max_{\mathbf{x}\in\mathcal{X}} \mathbf{LLM}(Q; \mathcal{D}, P_t^{\text{PP}}(\hat{\mathbf{x}}_{t-1}, S_{t-1})), \tag{1}$$

where the prompt $P_t^{\text{PP}}(\hat{\mathbf{x}}_{t-1}, S_{t-1})$ in iteration t involving previous iteration decisions $\hat{\mathbf{x}}_{t-1}$ and satisfaction factor $S_t = \frac{f(\hat{\mathbf{x}}_{t-1}^*, \mathbf{x}'; \mathbf{w}) - f(\hat{\mathbf{x}}; \mathbf{w})}{f(\hat{\mathbf{x}}; \mathbf{w})}$. Satisfaction factor S_t is defined as the relative gap between the optimized QR-obj function value and the original value prior to optimization. It also enables LLM to identify the most relevant $\hat{\mathbf{x}}$ and achieve a higher value of S_t. For convenience of handling, $\mathbf{x}'$ are enforced to be equal to historical values $\bar{\mathbf{x}}'$ and are expressed as equality constraints in the following steps. After that, a code safeguard checks the LLM-generated constraints code to see if it complies with the solvers' syntax.

Step 4: E2E Model. The E2E program $\min_{\mathbf{x}\in\mathcal{X}} g(\mathbf{x}, \mathbf{w})$ (see details in the Section 4) includes both decision variables for a specific problem and related features and parameters, called covariates. The E2E model includes two parts: (i) An offline trained Random Forest (Figure 2 ④) with a specific data predictor such as traffic congestion factors or operational cost of EBS. (ii) MIP: Following an earlier work (Biggs et al. 2022), we encode the Random Forest tree structure as an MIP. The objectives of the MIP model include both operational objectives (identical with the RF data predictor) and query-related QR-objs $f(\mathbf{x}; \mathbf{w})$ provided by QR-obj generators. In practice, the realization of query-relevant goal should not lead to unnecessary costs or compromise the original operational goals. Therefore, the optimization of the Query-relevant objective $f(\mathbf{x}; \mathbf{w})$

will be tackled as a secondary priority, following the assurance of the optimality of the original objective in MIP. Additionally, the parameterized variables $\mathbf{x}'$, in the form of equality constraints, are incorporated into the MIP constraints. Then the agent proceeds with the following model to achieve scoped-down optimal solutions:

$$\hat{\mathbf{x}}_t^* = \arg\min_{\hat{\mathbf{x}}} g(\hat{\mathbf{x}}; \bar{\mathbf{x}}', \mathbf{w}). \tag{2}$$

By parameterizing irrelevant decisions, the scoped-down problem $\min_{\hat{\mathbf{x}}} g(\hat{\mathbf{x}}; \bar{\mathbf{x}}', \mathbf{w})$ achieves efficient, optimal solutions $\hat{\mathbf{x}}_t^*$ in less computational time.

Step 5: Response Prompter. The E2E model iteratively generates optimal solutions until the user's query is satisfied (S_t). The satisfaction rate, which must be higher than zero and consistent across multiple iterations, controls the termination condition. Finally, the Response Prompter converts the optimal solutions and scores $(\mathbf{x}^*, S^*)$ into a more easily comprehensible form for the end-users.

Pseudocode Summary. We summarize the pseudocode of the City-LEO agents as follows: The pseudocode outlined in Algorithm 1 encapsulates the operational flow of the City-LEO agent, designed to process users' queries input within an urban operations management context. The input comprises the natural language user query Q, an initial empty set $\mathbf{x}_0'$ of parameterized decisions, and a zero-initialized satisfaction score S_0, along with the LLM prompts P^{IG} and $P^{\text{PP}}(\mathbf{x}_0', S_0)$. Initially, the Problem Matcher identifies an agent specific to the query's context. The QR-obj generator then starts, performing three critical tasks: it generates a query-relevant QR-obj function $f(\mathbf{x}; \mathbf{w})$ via the LLM, a satisfaction factor S_t, and produces safeguarded objective function code. The iterative process continues until either the maximum number of iterations is reached or the satisfaction factor becomes positive value (in the case of maximizing the QR-obj). Within each iteration, the problem tailor invokes empirical decision-related data $\mathcal{D}$ from a database and updates the LLM-Optimal parameterized decisions $\mathbf{x}_t'$. Subsequently, the algorithm solves an f-objective embedded E2E programming $\min_{\mathbf{x}\in\mathcal{X}} g(\mathbf{x}; \mathbf{w})$ that encapsulates the objective function f. It refines the solution space and updates both the satisfaction score S_t and the LLM prompts $P^{\text{PP}}(\mathbf{x}_0', S_0)$ for the next iteration. Upon convergence or fulfillment of the termination criteria, the algorithm outputs the optimized decision vector $\mathbf{x}^*$ alongside the final satisfaction score S^*, marking the end of the urban-centric City-LEO agent.

Algorithm 1 City-LEO Agent Pseudocode

Input: Query Q, initial value $S_0 = 0$, LLM prompts $P^{\text{IG}}, P_0^{\text{PP}}(\mathbf{x}_0', S_0)$.
Problem Matcher: Determine an area-specific agent.
QR-obj Generator:
1. Query-relevant QR-obj function $f(\mathbf{x}; \mathbf{w}) \overset{\text{def}}{=} \mathbf{LLM}(Q; P^{\text{IG}})$ generation.
2. Objective $f(\mathbf{x}; \mathbf{w})$ code generation (code safeguard checked).
3. Satisfaction factor $S_t = \frac{f(\hat{\mathbf{x}}_t^*; \bar{\mathbf{x}}', \mathbf{w}) - f(\bar{\mathbf{x}}; \mathbf{w})}{f(\bar{\mathbf{x}}; \mathbf{w})}$ generation.

while $\{t \leq \text{max-iterations}\}$ or $\{S_t > 0\}$ **do**
a. Problem Tailor:
1. Call decision-related empirical data $\mathcal{D} \overset{\text{def}}{=} (\bar{\mathbf{x}}, \bar{\mathbf{w}})$ from a database.
2. Update the LLM-Optimal parameterized decisions $\mathbf{x}_t'$:
$$\mathbf{x}_{t+1}' = \arg\ \max_{\mathbf{x}\in\mathcal{X}} \mathbf{LLM}(Q; \mathcal{D}, P_t^{\text{PP}}(\mathbf{x}_t', S_t)),$$
b. Solve a E2E model $g(\mathbf{x}; \mathbf{w})$ embedded with QR-obj f:
1. Optimal Scoping down solutions:
$$\hat{\mathbf{x}}_{t+1}^* = \arg\min_{\hat{\mathbf{x}}} g(\hat{\mathbf{x}}; \bar{\mathbf{x}}_{t+1}', \mathbf{w}),$$
2. Update $S_{t+1}, P_{t+1}^{\text{PP}}(\mathbf{x}_{t+1}', S_{t+1})$.
return $\hat{\mathbf{x}}_{t+1}^*, S_{t+1}$.
end while
return $\{\mathbf{x}^*\}\{S^*\}$.

3.1 Case: E-bike Sharing Operations

To facilitate the understanding of the aforementioned agent framework, this section presents a practical example of how the proposed City-LEO can be applied in a real-world EBS system. We specifically introduce

Prompt (Problem matcher and QR-obj generator):
Based on the **{INPUT *Q*}**, you need to extract the domain of the transportation system, and the desired QR-obj function to model the intention of uers' query.
The response format must be **domain**-[], **QR-obj code**-[], **SQL query**-[], where 'domain' is used to specify the field of the transportation system, for example, "bike-sharing system"; "QR-obj code" is used to describe the desired intention of the query, and follows the Gurobi syntax. For instructions, see the instance below:

***Instructional* Q&As:**
Question: How to reduce the percentage of low-powered e-bikes at parking spots No.1–NO.5 of the e-bike sharing system?
Answer:
Domain-[E-bike sharing tools];
QR-obj code-
[model.setObjective(gp.quicksum(model.getVarByName(f'u_{i}_{k0}') for i in range(1:5)), GRB.MINIMIZE)];
SQL query-[*SELECT* content, calculate_similarity(content, 'Query input') *AS* similarity_score
\ *FROM* documents\ *ORDER BY* similarity_score *DESC*].

Natural language query *Q* input:
How to make parking spots NO.1 and NO.2 have more bikes available for consumers to ride?

Thought-task breakdown-1
> Enter new AgentExecutor chain.
Task: generate QR-obj functions based on selected factors.
Action: **a**. invoking 'code generation' with 'increase available bikes at NO.1, NO.2 parking spots'; **b**.invoking 'code safeguard' with 'model.setObjective(...) '

Thought-task breakdown-2
Task: Check the factors related to the bike's availability.
Action: invoking: 'SQL generation' with 'Examines similarities between factors related to available bikes. '

Output:
>Finish chain.
Domain-[bike-sharing system];
QR-obj code-[model.setObjective((gp.quicksum(gp.quicksum(model.getVarByName(f'u_{i}_{k}') for k in K if k > k_0)/ C[i] for i in range(2:3))) , GRB.MAXIMIZE)];
SQL query-[SELECT content, calculate_similarity(content, 'Query input') AS similarity_score
\ FROM documents\ ORDER BY similarity_score DESC].

Figure 3: Prompt in Problem Matcher and QR-obj Generator

prompts with the Q&A instructions adopted from our City-LEO (as shown in Figure 3 and Figure 4). The query Q pertains to operational-level pursuits commonly raised by EBS managers. We use $K = \{k_1, k_2, k_3\}$ to represent the set of discretized states of charge (SOC) levels of batteries, that is, k_1, k_2 and k_3 denote low SOC level (e.g. lower than 30%), medium SOC level (e.g. 30% - 70%) and high SOC level (e.g. higher than 70%), respectively. Typical pursuits include reducing the number of low-powered bikes and adding available bikes at certain locations. In particular, Figure 3 shows the City-LEO agent working on a real-world operational problem and a user's question about EBS rebalancing. This interaction involves both understanding and generating responses based on specific technical and domain-specific requirements. Next, we will elaborate on the user's query, prompt design, and thought-task breakdown as follows:

User's Query: The user, often an EBS manager, adopts City-LEO to inquire about operational decisions for increasing the number of available bikes at parking spots NO.1 and NO.2 (query Q) while ensuring the lowest possible operating cost. The crux of this matter lies in accurately defining "available bikes," leveraging operational data from the database in an automated manner, and identifying the most pertinent locations with NO.1 and NO.2 parking spots to re-optimize the EBS system.
Prompt (Problem Matcher and QR-obj Generator): The prompt first instructs LLM on how to extract the area-specific APIs (domain of the transportation system) and generate the desired QR-obj function from the user's query. The output response is specifically formatted as a "domain" and "QR-obj function codes" based on the prompt line (prompt matcher and QR-obj generators), as shown in Figures 3 and 4. In addition, we also provide *instructional Q&A* prompts to improve the precision of generating SQL queries, domains, and codes of QR-obj functions. In our case study, the given prompt specifies the domain as "EBS system," and the QR-obj code outputs a Gurobi solver code snippet. This code snippet can be utilized to optimize the number of available e-bikes at particular parking spots.

Thought Process Breakdown (Problem Matcher and QR-obj Generators): Based on the input and prompt, City-LEO breaks down the query into sub-tasks following a step-by-step process: First, City-LEO analyzes the relevant factors associated with "available bikes" recorded in the database through LLM-generated SQL. These factors include historical records of operational-level decisions or parameters, such as rebalancing bikes and initial allocations. Secondly, City-LEO generates an QR-obj function and corresponding Gurobi code snippets (e.g., `model.setobjective`(gp.quicksum(model.getVarByName(f'u_{i}_{k}') for i in range(55) for k in [1,2]), GRB.MAXIMIZE), see section 4 for notation details). This QR-obj function quantifies the number of "available bikes" using predetermined parameters in the initial stage. Subsequently, the code safeguard further guarantees the successful execution of the generated code in the E2E dispatch optimization model.

Prompt (Problem tailor):
Based on **{INPUT *Q*}**, you need to extract the specified problem scope, create SQL instruction code, and choose the most relevant subset of decisions. The remaining subset is parameterized as a historical record.
The response format must be **scope size**-[], **SQL query**-[], **parameterized constraints**-[], where "Scope size" refers to the subsets of queries related to decision-making in the E2E optimization problem. "SQL query" refers to the extraction instructions for obtaining historical data. "Constrinats generations" refers to the code that defines constraints using the Gurobi syntax.

***Instructional* Q&As:**
Question: How may the percentage of low-powered vehicles at parking spots 1–5 of the E-bike sharing system be reduced?
Answer:
Scope size-[i in range(1:5)];
SQL query-[*SELECT* station_id, COUNT(*) *AS* low_battery_bikes_count\ *FROM* bikes\ *WHERE* battery_level < 20\ *GROUP BY* station_id\ HAVING station_id *IN* (1, 2, 3, 4, 5)\ *ORDER BY* low_battery_bikes_count *DESC*];
Parameterized constraints-[model.addConstr(u[i] == d[i], for i in range(4:29)].

Natural language query *Q* input:
How to make parking spots NO.1 and NO.2 have more bikes available for most consumers to ride?

Thought-task breakdown-1
> Enter new AgentExecutor chain.
Task: Check the most relevant parking spot with NO.1 and NO.2 parking spots based on historical records of available bikes.
Action: invoking: **a**. 'SQL generation' with 'Examine records of **x,w** related to available bikes at NO.1 and No.2 spots. '; **b**. 'Analyses the most pertinent parking spots with NO.1 and No.2 parking spots based.'

Thought-task breakdown-2
Task: generate constraints code based on the selected parking *spots*.
Action: invoking 'code safeguard' with 'model.addConstrs(...) '.

Output:
>Finish chain.
Scope size-[i in range(1:5)];
SQL query-[SELECT station_id, COUNT(*) AS low_battery_bikes_count\ FROM bikes\ WHERE battery_level < 20\ GROUP BY station_id\ HAVING station_id IN (1, 2, 3, 4, 5)\ ORDER BY low_battery_bikes_count DESC]; **Constrinats generations**-[model.addConstr(u[i] == d[i], for i in range(4:29)].

Figure 4: Prompt in the Problem Tailor

Output: The model's output consists of the domain "bike-sharing system" and codes for a QR-obj function aligned with Gurobi syntax. The QR-obj function, in particular, is meant to maximize the number of e-bikes with at least k_1 level state of charge (SOC) at No.1 and No.2 parking spots.

Prompt (Problem Tailor): After the area-specific agents have been matched, the users' inquiry is processed by the problem tailor. The prompt of problem tailor consists of three components: historical data retrieval, problem scoping, and constraint generation. First, the prompt provides instructions on how to use SQL retrieve query-relevant operational information that is stored in the database. The information, such as the shared e-bike travel matrix and the redistribution of bikes among the spots, enables LLM to analyze the most relevant parking spots or decisions with queries.
For instance, Figure 4 illustrates an example of an instructional Q&A session on decreasing the proportion of low-powered vehicles at specified EBS parking spots. Secondly, prompts instruct on problem scoping, specifically in this context, to aid in the selection of EBS parking spots that are relevant to the query. For example, parking spots ranging from No. 4 to No. 8 have a higher frequency of trips and a more frequent record of rebalancing operations from No. 1 and No. 2 parking spots. This helps to scope down the search area from $\mathcal{X}$ to a specific subset consisting of parking spots No. 1, No. 2, and No. 4 to No. 8 in the E2E dispatch optimization problem.

Thought Process Breakdown (Problem Tailor): Guided by the prompt of problem tailor, City-LEO then breaks down the user query into tasks including a query-relevant information search and constraint generation. In line with the task breakdown suggested in the QR-obj generator, the initial step involves retrieving relevant information, such as operational-level decisions or parameters, from a database using SQL. This information facilitates the process of determining the scope of a optimization problem and generating Gurobi constraints code (i.e., `model.addConstr()`).

Output: The final output is a combination of SQL code, QR-obj and Gurobi optimization formulation tailored to the specifics mentioned in the query. This output is intended to facilitate decision-making that enhances bike availability based on historical usage patterns.

4 End-to-End Model for the EBS System

Having outlined the algorithm flow of City-LEO in Section 3, we next focus on the E2E model for the EBS system. EBS is an effective and appealing means of last-mile transportation by offering a convenient and flexible travel mode. However, the operators of the EBS system face challenges in dealing with the imbalance

between supply and demand (Jin et al. 2023), and recharging depleted batteries for e-bikes. To handle the operations management problems of urban EBS systems, in this section, we propose the optimization models embedded in the LLM agent that facilitate an efficient and transparent decision-making process. First, we construct the conventional deterministic optimization model for rebalancing e-bikes and battery swapping. Then we formulate the RF-based E2E optimization model, and leverage the feature-to-decision framework to obtain more accurate and interpretable solutions. In addition, to cope with diverse queries flexibly and reduce computational time, we explain how the LLM-based agent redefines the management problem and scopes down the size of the E2E optimization model.

4.1 The E-bike Rebalancing and Battery Swapping Model

Consider the EBS system with multiple parking spots, we construct an integer program to optimize the rebalancing and battery swapping strategies for the next several hours. At the initial of the horizon, two types of staff drive different vehicles to distribute e-bikes among multiple parking spots, and to replace depleted batteries with fully charged ones, respectively. Let $I = \{1, \cdots, |I|\}$ denote the set of parking spots. To ensure service quality, e-bikes with low SOC are not available for satisfying users' demands. Before rebalancing and battery switching, the initial number of e-bikes with SOC k ($k \in K$) at parking spot i is v_{ik}, and the capacity of parking spot i is C_i. The capacity of the vehicle for carrying e-bikes is N, and the initial number of fully charged e-bikes on the dispatching vehicle is r_{k_3}. Other staff carries fully charged batteries to replace the depleted batteries, and the maximum number of fully charged batteries that the staff carries when setting off is B. After dispatching e-bikes and swapping batteries, the stock of e-bikes with SOC k at each parking spot i is u_{ik}. At the beginning, the operator has to decide the number of batteries y_{ik} that are replaced by fully charged batteries at spot i, and the number of e-bikes z_{ik} that are moved out or dropped off at spot i with SOC k. In addition, let c_1 and c_2 denote the unit cost for swapping batteries and moving e-bikes, respectively. Define the customers' net demand at spot i is ξ_i, and since customers always prefer to use e-bikes with high SOC, we add penalties p_1 for using e-bikes with medium SOC level to satisfy traveling demands. Penalties p_2 are for unmet demands σ_i. Given the notation, we formulate the following deterministic model to optimize the rebalancing and battery swapping (RBS) decisions:

$$\text{(RBS)} \quad \min_{\mathbf{y},\mathbf{z},\mathbf{u}} \quad \sum_i \left(\sum_{k<k_3} c_1 y_{ik} + \sum_{k\in K} c_2 |z_{ik}| \right) + p_1 \sum_i (\xi_i - \sigma_i - u_{ik_3})^+ + p_2 \sum_{i\in I} \sigma_i \tag{3}$$

$$\text{s.t.} \quad u_{ik} = v_{ik} - y_{ik} + z_{ik}, \quad \forall i \in I,\ k < k_3, \tag{4}$$

$$u_{ik_3} = v_{ik_3} + \sum_{k<k_3} y_{ik} + z_{ik_3}, \quad \forall i \in I, \tag{5}$$

$$\sum_{k>k_1} u_{ik} + \sigma_i \geq \xi_i, \quad \forall i \in I, \tag{6}$$

$$\sum_{k\in K} (v_{ik} + z_{ik}) - \xi_i \leq C_i, \quad \forall i \in I, \tag{7}$$

$$\sum_{i\in I} \sum_{k<k_3} y_{ik} \leq B, \tag{8}$$

$$r_{k_3} - \sum_{i\in I} \sum_{k\in K} z_{ik} \leq N, \tag{9}$$

$$r_{k_3} - \sum_{i\in I} z_{ik_3} \geq 0, \tag{10}$$

$$\sum_i z_{ik_3} \geq 0, \tag{11}$$

$$\sum_i z_{ik} \leq 0, \quad \forall k < k_3, \tag{12}$$

$$u_{ik} \geq 0, \quad \forall i \in I, \forall k \in K;\ y_{ik} \geq 0, \quad \forall i \in I, \forall k \in K \backslash k_3. \tag{13}$$

Objective (3) represents the total operational costs for swapping depleted batteries, dispatching e-bikes, and the penalties for unmet demands and not providing e-bikes with high SOCs to satisfy demands. Constraints (4) and (5) are the inventory-energy balance equations, that is, the stock of e-bikes with SOC k at parking spot i is equal to the stock in the previous period minus the number of replaced batteries and then plus

the net dispatched e-bikes. Constraint (6) represents the relationship between the stock of e-bikes and the users' demands. Constraint (7) imposes space limits to make sure that the stock of e-bikes do not exceed the capacity of the parking spot. The initial number of fully charged batteries carried by the staff is $\sum_i \sum_{k<k_3} y_{ik}$, and constraint (8) sets the upper limit B for the number of swapped batteries. Constraint (9) means that the number of e-bikes on the vehicle does not exceed the capacity limit N. Constraint (10) indicates that the total number of dispatched e-bikes is less than the initial number of fully charged e-bikes on the operator's vehicle. Constraint (11) indicates that the total number of fully charged e-bikes dispatched among the parking spots is nonnegative. Constraint (12) means that the total number of e-bikes with low SOC and medium SOC will not increase after rebalancing and battery swapping operations. Constraint (13) ensures the nonnegativity of the decision variables.

4.2 RF-based E2E Model

Considering the integration of OR models and LLM, the deterministic optimization problem (RBS) has some limitations. First, deterministic optimization cannot handle users' stochastic travel demands, which may result in suboptimal or even infeasible solutions. Second, the deterministic model leads to low data utilization and lacks the flexibility to efficiently respond to the diverse queries of users. Therefore, extending the approach of (Biggs et al. 2022), we construct the following RF-based E2E model to involve more features and improve the utilization of historical data. In addition, the tree-based E2E optimization mitigates the error accumulation of predicting stochastic parameters, and avoids repeatedly training the RF models every time a user submits a query. The tree structure is also suitable for describing unknown and complex interactions of the covariates (e.g. features and variables) and objectives, and enhances the interpretability and transparency of the decision-making process.

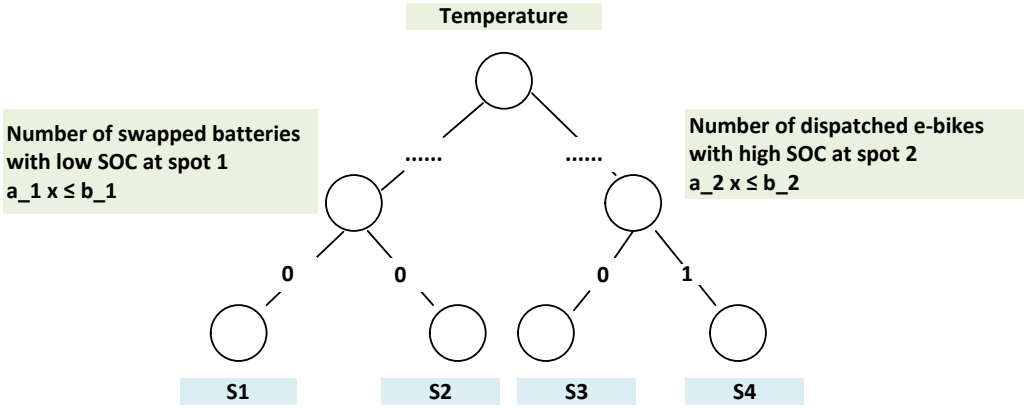


Figure 5: Sample Tree of the Trained Random Forest

For the EBS system, we use Figure 5 to depict a sample tree of the RF, which is trained by the historical meteorological data and operational records. The tree structure can reflect the inherent uncertainty of users' demand. Suppose the number of trees in the RF is $|H|$, and the leaf node of each tree h represents the predicted operational cost $R(\mathbf{x}, \mathbf{w})$, where $\mathbf{x} = (\mathbf{y}, \mathbf{z})$ is the decision variable and $\mathbf{w}$ represents the features. Our goal is to minimize the cost for rebalancing and battery swapping by minimizing $R(\mathbf{x}, \mathbf{w})$. Let N^h denote the number of nodes (except the leaves) in tree h. For each interior node, let p_i, l_i and r_i be the immediate parent, the left child and the right child, respectively. We use L^h to represent the set of leaves in tree h, and use R_j^h $(j \in L^h)$ to denote the score of each leaf. Given the trained RF, we introduce binary variables q_{ij}^h to select branches and decide the range of the variables, and the corresponding 0-1 mixed integer programming is as follows:

$$\text{(RBS-E2E)} \quad \min_{\mathbf{x},\mathbf{q},\mathbf{u}} \quad \frac{1}{H} \sum_{h=1}^{H} \sum_{j \in L^h} R_j^h q_{p_j,j}^h \tag{14}$$

$$\text{s.t.} \quad a_{i,h}\mathbf{x} - M\left(1 - q_{i,l_i}^h\right) \leq b_i^h, \forall h \in H, i \in N^h, \tag{15}$$

$$a_{i,h}\mathbf{x} + M\left(1 - q_{i,r_i}^h\right) \geq b_i^h, \forall h \in H, i \in N^h, \tag{16}$$

$$q^h_{i,l_i} + q^h_{i,r_i} = q^h_{p_i,i}, \forall h \in H, i \in N^h, \tag{17}$$

$$\sum_{i \in L^h} q^h_{p_i,i} = 1, \forall h \in H, \tag{18}$$

$$q^h_{i,l_i}, q^h_{i,r_i}, q^h_{p_i,i} \in \{0,1\}, \forall h \in H, i \in N^h, \tag{19}$$

Constraints (4)-(5), (8)-(13).

The objective (14) is to minimize the predicted cost, that is minimize the score of the selected leaves. Constraint (15) and constraint (16) are the big-M reformulations of logical constraints and determine which leaf the solution $\mathbf{x}$ lies in. Constraint (17) ensures that if a parent node is inactive, its children must also be inactive; but if any child is active, then the parent must be active as well. Constraint (18) guarantees that within each tree h, one and only one leaf can be active. Constraint (19) defines the binary variables. According to the model (RBS), the EBS system also needs to satisfy constraints (4)-(5) and (8)-(13) to ensure capacity limitations, inventory-energy balance equations and so on.

Compared with the deterministic optimization model (RBS), the model (RBS-E2E) integrates a random forest with MIP to involve more environmental features and uncertainties. The hierarchical tree structure can reflect the decision path from the root to a leaf node, which promotes the transparency and interpretability of the decision process. However, the model (RBS-E2E) lacks the flexibility to accommodate operators' diverse requirements in practical implementation. LLM offers a promising path to enhancing accessbility through conversational interactions. Therefore, we will introduce LLM-embedded E2E model in the subsequent section.

4.3 LLM-embedded E2E Model

The integration of the LLM agent and RF-based E2E optimization can lead to mutual enhancement for practical EBS management. According to the users' queries, the LLM agent utilizes its powerful reasoning ability and generates a new objective for the MIP. Users' queries usually focus on the operations of EBS system in various urban areas, and are less likely to involve the entire city. To cope with users' queries flexibly and avoid solving large-scale optimization models repeatedly, LLM contributes to scoping down the optimization problem to reduce computational complexity without significantly compromising the accuracy of the solutions. Specifically, after processing the users' queries, the LLM agent screens out the parking spots $\hat{I}$ and variables $\hat{\mathbf{x}} = (y_{ik}, z_{ik})$ $(i \in \hat{I})$ that are most relevant to the query, and then parameterizes the other decision variables $\mathbf{x}' = (y_{ik}, z_{ik})$ $(i \in I \backslash \hat{I})$ with the mean values of the historical data $\bar{\mathbf{x}}'$. For example, if a user is concerned about the operations of parking spots No.5-No.10, based on geographical distance and historical trip records, we can select some spots that are most relevant to the six spots (that is, the set of spots $\hat{I}$), rather than optimizing the EBS operations of the entire city. Figure 6 represents that the variables associated with spot No.1 are not closely relevant to the user's problem. On the whole, the LLM-embedded optimization tool utilizes RF-based E2E optimization to learn from data and generate new decisions, and then provides responses to users through conversational dialogues. Define $A(\mathbf{w})F(\mathbf{x}) \leq m(\mathbf{w})$ to represent the feasible region $\mathbf{x}$. After scoping down the model (RBS-E2E), we have the following optimization program:

$$\text{(RBS-LLM)} \quad \min_{\hat{\mathbf{x}},\mathbf{q},\mathbf{u}} \quad \frac{1}{H} \sum_{h=1}^{H} \sum_{j \in L^h} R^h_j q^h_{p_j,j} \tag{20}$$

$$\min_{\hat{\mathbf{x}},\mathbf{q},\mathbf{u}} \quad \mathbf{LLM}(Q; \mathcal{D}, P^{\text{PP}}) \tag{21}$$

$$\text{s.t.} \quad A(\mathbf{w})F(\hat{\mathbf{x}}, \bar{\mathbf{x}}') \leq m(\mathbf{w}), \tag{22}$$

Constraints (17)-(19).

Objective (21) is generated by the LLM agent based on the user's query Q. To maintain minimum operational cost (20) and then accommodate user's requirement, we set a lower priority to objective (21). In the multi-objective optimization problem, we solve the programming model with objective (20) first, and then re-optimize the problem with the newly added objective (21) while maintaining the first objective optimal. Constraint (22) represents the cluster of constraints about selecting branches of the trees, rebalancing e-bikes and battery swapping operations. Constraints (17)-(19) involve binary variables to explain how to select branches and leaves in the random forest.

After scoping down the full-scale optimization problem and parameterizing some redundant variables $\mathbf{x}'$, the LLM-embedded E2E problem with shorter computational time is more flexible and transparent to cope with

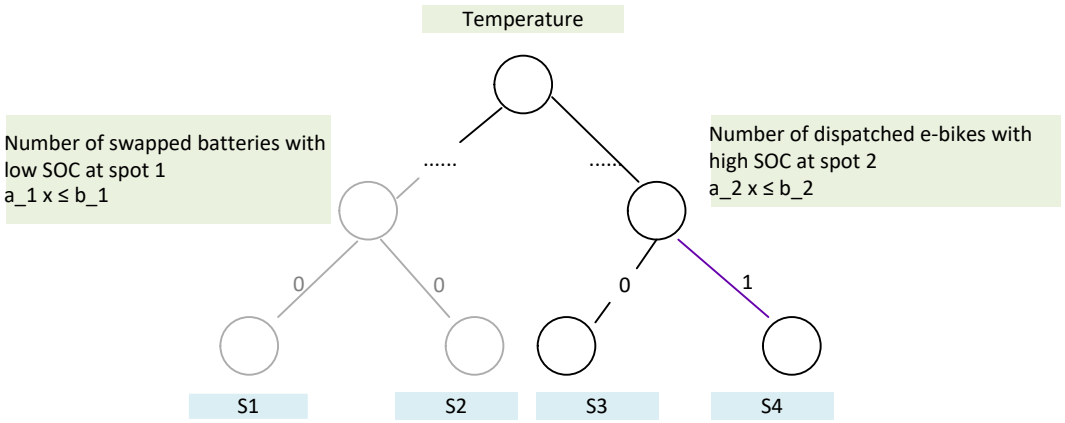


Figure 6: Sample Tree of the Trained Random Forest after Scoping Down the Problem

users' queries. In addition, the feasible region of model (RBS-LLM) gets smaller, and we will compare the optimality gaps and computational time in the case study.

5 Case Study

In this section, we demonstrate the effectiveness of the City-LEO agent with a case study of the Pronto EBS system in Seattle, USA. Specifically, Section 5.1 offers a brief description of the data and parameter settings. Section 5.2 introduces the experimental setup. Sections 5.3, 5.4, and 5.5 present numerical results about relevance evaluation and accuracy evaluation of City-LEO, compared with Gurobi. Furthermore, we assess the scope-down policy of City-LEO in the supplemental experiment outlined in Appendix D.

5.1 Data Description and Parameter Settings

We adopt the real-world case of the Pronto Cycle Share system in Seattle. The system comprises 55 parking spots situated in the major neighborhoods of the city, with a fleet of 500 bicycles. Cycle Share Dataset (Kaggle 2020) provided by Pronto contains data on the parking spots, trips, and weather information from October 13, 2014 to August 31, 2016. The trip records contain rental and return data, traveling distance, and users' information. The weather dataset encompasses daily weather-related indicators, such as temperature, humidity, and visibility. Based on the historical trip records and the stock information at parking spots, we generate data about e-bikes dispatching and battery swapping operations through simulation and ensure inventory-energy balance. Furthermore, sufficient historical data enhances City-LEO's reasoning ability for supplementing prior knowledge and generalization ability of E2E model.

We analyze 283,143 pieces of historical trip records spanning from October 2014 to August 2016, and focus on the operations management of the EBS system during the noon hours, when the users are more active than the other periods. The capacities of vehicles for carrying e-bikes and batteries are 60 and 200, respectively. Assume that the initial number of e-bikes with high SOCs on the vehicle is $r_{k_3} = 60$, and the initial number of fully charged batteries on aboard the vehicle is $B = 142$. Regarding the operational costs, the unit cost for swapping depleted batteries or relocating e-bikes is \$1. In order to reduce the range anxiety and enhance the service quality, we set $p_1 = 0.4$ to penalize using e-bikes with medium SOC level to satisfy traveling demands, and the penalty parameter p_2 for stock-out of available e-bikes is \$0.8. The cruising range of a high SOC level e-bike is approximately 50km-80km (China Academy of Urban Planning and Design 2023), and the average power consumption per trip is about 3%. Therefore, SOC changes for each traveling trip is negligible. To validate the effectiveness of City-LEO across diverse datasets, we have further developed two additional datasets derived from Cycle Share Data. According to geographical proximity, we cluster the 55 parking spots into 20 and 35 clusters, respectively. Due to the page limit of the paper, we have omitted the detailed experimental results pertaining to the two datasets. The results consistently indicate

that City-LEO has superior performance in terms of relevance and accuracy when benchmarked against the full-scale optimization model.

To train the random forest in the E2E optimization model, we import the historical trip data, geographic information of parking spots, weather information, and operational records into the LLM database, and set 80% of the 8924 pieces of data as the training set, and the 20% of the data is used for the test set. The maximum number of trees in a random forest is 100, and the maximum depth of trees is 400. The accuracy of random forest is 93.92% on the training set and 58.22% on the test set.

5.2 Experimental Setup

The goal of the case study is to assess City-LEO's ability to improve *relevance* without excessively compromising *accuracy* in resolving queries regarding EBS management. We conduct two types of numerical experiments, namely *relevance* and *accuracy* tests, in order to validate the advantages of City-LEO.

In both experiments, we employ the unit-test approach, which is frequently utilized in software development, to assess the performance of City-LEO (Li et al. 2023a). We select three representative operational management scenarios within the EBS system, which comprehensively cover the perspectives of EBS operators, customers, and government. For each scenario, we propose a set of 19 queries accompanied by human-labeled ground-truth answers (see Appendix B.1). To cope with the variability of the LLM's output, we conduct five experiments for each query and take the average values of the relevancy and accuracy assessments.

We conduct relevancy and accuracy tests using GPT-4.0 and construct an agent framework based on Langchain (a typical framework for developing LLM-driven applications). In addition, we solve the MIP in the E2E model using in Gurobi 10 on a macOS system with an Apple M1 Pro CPU and 16GB RAM.

5.3 Relevance Evaluation

The objective function related to the user's query is proposed by utilizing LLM's reasoning ability on the input prior knowledge. To demonstrate the effectiveness of the generated objective function, we perform relevance tests to assess the similarity between the objective functions generated by City-LEO and ground-truth expressions labeled by human.

In our comparative experiments, we not only scrutinize the expressions of the generated objectives, but also assess whether the generated function aligns with the ground-truth formulation in terms of mathematical essence. Sometimes the codes of the two functions are different, yet their mathematical implications remain identical. For example, the following two different codes both represent maximizing the total number of available e-bikes after dispatching and battery swapping operations.

- `model.setObjective (gp.quicksum (model.getVarByName (f'u_{i}_{k}') for i in range(55) for k in [1, 2]), GRB.MAXIMIZE)`
- `model.setObjective (gp.quicksum (gp.quicksum (model.getVarByName (f'u_{i}_{k}') for k in K if k >= k_2) for i in I), sense = GRB.MAXIMIZE)`

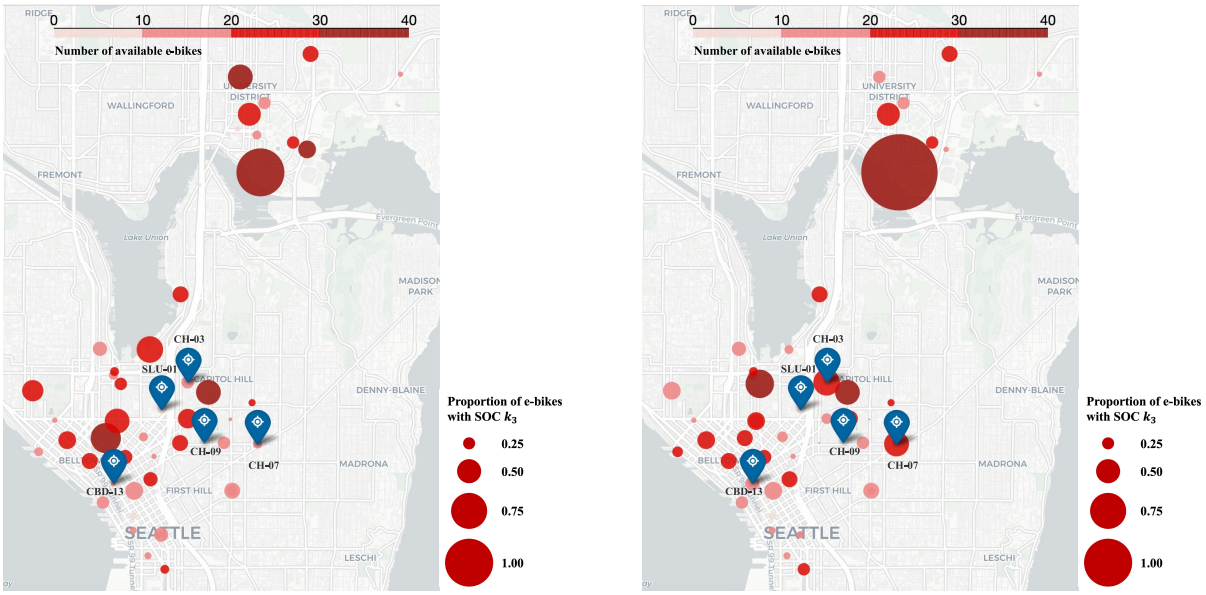
To this end, we propose two measures based on Jaro-Winkler distance, *text similarity* and *results similarity*. The two metrics evaluate the similarity between the "generated objective function" and the "ground-truth objective function" in terms of codes and mathematical essence (see Appendix B.2 for details), respectively. Next, we analyze the performance of City-LEO for generating query-related objective functions under the two metrics.

Table 1: Results of the Relevance Test

Prompts[1]	In-sample test		Out-of-sample test	
	Result similarity	Text similarity	Result similarity	Text similarity
0	——	——	0.61	0.90
3	0.89	0.96	0.73	0.93
6	0.91	0.96	0.78	0.94
9	0.90	0.97	——	——

[1] The term "Prompts" refers to the number of Q&A instructions used in each experiment.

Table 1 illustrates that City-LEO performs well in out-of-sample evaluation test in which no instructions are given, with a similarity of 0.61 and 0.90 with the 0 prompt. Increasing the number of prompt sets (as



(a) Distributional performance of FULL.

(b) Distributional performance of City-LEO.

Figure 7: Spatial Distribution of EBS Operations Performance

shown in "Prompts" in Table 1) mildly improves the similarity of the generated objective to the ground-truth function, both in the in-sample tests and the out-of-sample tests. The results demonstrate that City-LEO effectively leverage LLM's reasoning and generalization capabilities on prior knowledge to propose objective functions, which better match the users' queries and can be added to the E2E optimization model. With effective query-related objective, next we will investigate whether City-LEO has advantages in redefining and solving neighborhood-level mathematical programs, compared with the full optimization model for the whole city.

5.4 Performance of City-LEO Operations

In this section, we use City-LEO to enhance the quantity of available e-bikes, as well as increase the proportion of e-bikes with higher SOC, at the five designated parking locations shown by the blue symbols in Figure 7. We denote the baseline model as "FULL," representing the E2E optimization model without any scoping-down operations and aiming at optimizing the dispatch and battery swapping operations within the whole city. Figure 7 shows the spatial distribution of available e-bikes with SOC levels above k_2 and k_3 (size of dots), as well as the proportion of e-bikes at the k_3 SOC level (the varying shades of dots). More precisely, the bigger and darker shade of the dots indicates the presence of a greater number of available e-bikes and proportion of e-bikes at the k_3 SOC level.

We find that City-LEO outperforms FULL in satisfying users requirements. For example, as depicted in Figures 7, for the five query-relevant parking spots, there are more available e-bikes with a higher SOC when adopting City-LEO (as shown in Figures 7(b)) to optimize dispatch and battery swapping strategies. The results highlight the practical value of employing City-LEO as an improved alternative to FULL in urban EBS operations.

5.5 Accuracy Evaluation

Compared with the FULL model, City-LEO aims to improve computational efficiency by inferring a scaled-down domain that is highly relevant to the user's query. For example, when a user submits a query regarding parking spots No.1-No.17 in an urban subregion, City-LEO needs to utilize the reasoning ability on the historical data to identify the parking spots that are most closely related to the operations of spots No.1-No.17 for optimization purposes. In the E2E optimization model, the decision variables associated with the less relevant parking spots are subsequently parameterized as the mean values of the historical data. As a result, our proposed method prioritizes the "scope" that is relevant to the users' queries, yet the optimization program offers suboptimal solutions compared to optimizing all the parking spots by the FULL model. Hence, it is

necessary to evaluate the discrepancies with the optimal solutions of the FULL model as well as computational efficiency.

City-LEO and the baseline FULL model share identical objectives, including the objectives regarding operational costs (referred to as "FULL-obj"), and the extra-generated query-relevant objectives (referred to as "QR-obj") in each experiment. We assess City-LEO's computational accuracy and effectiveness by examining two important deviations from the baseline model: "*global suboptimality*" and "*local satisfaction*," while also comparing CPU time. The metrics of "global suboptimality" and "local satisfaction" quantify the differences between the optimal solution of the baseline mode "FULL" and City-LEO in terms of the "FULL-obj" and "QR-obj" values, respectively (see Appendix B.3 for details).

Furthermore, we propose the metric "Locality" to measure the proportion of parking spots mentioned in customers' queries. For instance, if customers only care about 17 specific parking spots among 55 EBS locations in total, then the locality is 30.91%. And the decision variables associated with these 17 designated parking spots and their relevant locations (analyzed by the City-LEO) will not be parameterized. A higher level of locality indicates that the scoped-down problem is more similar to the FULL optimization model. In addition, we also conduct in-sample and out-of-sample tests to investigate the influence of prompts on the computational accuracy of City-LEO. Next, we will elaborate on our experimental results with query-relevant linear objectives and non-linear objectives, respectively.

5.5.1 Performance of City-LEO with Linear Query-relevant Objective.

As seen in Table 2, compared with FULL model, City-LEO exhibits superior computational efficiency under different locality settings. In addition, City-LEO also has higher Local satisfaction values and possesses advantages in meeting users' requirements, while maintaining lower global suboptimality and not significantly compromising computational accuracy. For example, when the locality is 40%, the user aims at optimizing the operations of 22 parking spots in a certain district of the city. Based on the geographic information and historical origin-destination information of the trips, City-LEO will screen out some parking spots that are operationally relevant to the 22 parking spots, and the variables associated with the other parking spots will be parameterized into historical values. The results show that the CPU time of City-LEO is about 16.26% faster than FULL on average, the *global suboptimality* is as small as 0.62% and the *local satisfaction gain* is 2.44% on average.

Table 2: Results of Accuracy Tests with Linear Query-relevant Objectives

Locality	In-sample test				Out-of-sample test			
	CPU time (s)		Global suboptimality	Local satisfaction gain	CPU time (s)		Global suboptimality	Local satisfaction gain
	City-LEO	FULL			City-LEO	FULL		
20%	168.56	195.64	0.71%	2.33%	154.96	179.17	0.98%	2.26%
40%	171.65	212.57	0.62%	4.08%	183.86	218.90	0.65%	7.18%
60%	180.97	212.83	0.59%	1.14%	179.91	209.45	0.72%	4.33%
80%	178.34	214.83	0.56%	2.22%	180.59	199.84	0.95%	1.27%

In addition, local satisfaction gain presents a comparison of "satisfaction levels" to users' requirements. For instance, in Table 2, the local satisfaction gains with varying locality values are all positive, which suggests that the EBS operation strategies provided by City-LEO better fulfill users' requirements compared to FULL. Furthermore, in the out-of-sample tests, the *global suboptimality* and *local satisfaction gain* are consistently maintained at lower levels, which suggests that City-LEO is less prone to overfitting when faced with various queries.

5.5.2 Performance on Nonlinear Query-relevant Objectives.

City-LEO sometimes generates convex nonlinear objective functions based on the intent of users' queries. For example, City-LEO generates $\sum_{i\in I}(\sum_{k\in K} u_{ik} - C_i)^2$ to denote the *utilization rate of parking spots* or $\sum_{i\in I}\sum_{k\in\{k_1,k_2\}}(x_{ik})^2$ to present the turnover rate of e-bikes with a low SOC. Embedding such a quadratic objective into the E2E optimization model often results in an inefficient computational process. In order to assess the performance of City-LEO with nonlinear objectives in out-of-sample tests, we offer two representative instances, namely "maximizing the *utilization rate of parking spots*" (denoted as "NLP-obj-1") and "minimizing the *turnover rate of e-bikes with low SOC* (denoted as "NLP-obj-2"), respectively.

Considering two different nonlinear objectives, Table 3 shows the comparison results of computational performances between City-LEO and FULL. Results show that City-LEO exhibits superior computational efficiency while incurring certain compromises on accuracy. For instance, with generated objective NLP-obj-1, City-LEO takes only 94.92s to solve the E2E optimization problem, while the FULL model fails to generate the optimal solutions within one hour (and the optimality gap is about 66%). We compare the optimal solutions provided by City-LEO with the feasible solutions generated by FULL after one-hour computation. When the user is concerned about 20% of the parking spots, the global suboptimality is 11.75%, and the local satisfaction gain -3.70%.

Table 3: Results of Accuracy Tests with Nonlinear Query-relevant Objectives

Locality	NLP-obj-1				NLP-obj-2			
	CPU time(s)/GAP(%)		Global suboptimality	Local satisfaction gain	CPU time(s)		Global suboptimality	Local satisfaction gain
	City-LEO	FULL			City-LEO	FULL		
20%	94.92	3600 (66.44%)	11.75%	-3.70%	85.21	520.99	11.96%	6.65%
40%	149.19	3600 (66.76%)	8.14%	0.07%	90.24	485.88	7.86%	4.88%
60%	468.64	3600 (66.16%)	5.23%	-4.35%	158.81	499.04	6.67%	-7.50%
80%	624.80	3600 (66.37%)	2.08%	-0.92%	315.18	486.25	4.03%	-4.32%

In terms of NLP-obj-2, we observe that City-LEO exhibits similar benefits in terms of CPU time. As the level of localization increases, City-LEO's Global suboptimality progressively decreases, e.g., from 11.96% to 4.03%. When the locality is 20% or 40%, the results indicate that City-LEO better satisfies the user's requirements. In summary, when implementing QR-obj with quadratic terms, City-LEO significantly outperforms the baseline FULL.

6 Conclusion

In this paper, we introduce City-LEO, an LLM agent integrated with E2E optimization for enhancing practical relevance and decision-making efficacy in an urban EBS management case. Such an agent is designed to bridge the gap between the intricate nature of OR tools and the practical needs of city operators (EBS operators in our context). It provides a conversational interface that realizes the effective accomplishment of query-relevant quantitative targets by flexibly scoping down large-scale optimization problems. It also offers responsible tools which are more user-friendly, transparent for city operators with little knowledge of operations research. Furthermore, by leveraging pre-trained E2E optimization models and incorporating stakeholder natural language query-relevant features, City-LEO also enhances the quality of solutions in uncertain urban decision environments.

In a real-world case, we examine the performance of our proposed City-LEO in terms of query relevance and optimality for large-scale problems. In particular, we observe that City-LEO shows reliable query-relevant objective generation, achieving 0.90 text similarity without prompts in out-of-sample test. The result ensures that subsequent E2E models obtain optimal solutions closely relevant to the user's query. In the accuracy test, City-LEO demonstrates better computational efficiency, particularly when dealing with nonlinear query-relevant objectives, and achieves lower levels of global suboptimality relative to the baseline model under various types of query-relevant objectives. The results indicate that our proposed City-LEO achieves greater relevance with users' requirements while incurring reasonably small compromise on optimality. At last, through adjusting the maximum permitted number of parameterized parking spots, City-LEO has been demonstrated to provide a reasonable range of parameterization operations to fulfill users' queries. The aforementioned experimental advantages essentially stem from the benefits of *scoping down* operations inside E2E model via City-LEO. LLM's reasoning capability enables City-LEO to make substantial progresses in reducing the complexity and intransparency, which are typically encountered in traditional OR tools used for urban operations management.

In addition, by facilitating a clearer understanding of the decision-making process, City-LEO helps align municipal decisions with community needs and enhances public trust in an interactive way. Furthermore, the outcomes of our study not only affirm the viability of using LLMs and E2E optimization in public administration but also reveal the potential for the technologies to transform city management into a more inclusive and transparent practice. While City-LEO significantly advances the state of the art, future work may explore integrating more dynamic data inputs and real-time decision-making capabilities to further enhance its responsiveness and accuracy. Additionally, extending the framework to incorporate feedback

loops from implemented decisions could refine its prescriptive accuracy and user satisfaction, creating a continuously improving system that adapts to the evolving urban landscape.

References

AhmadiTeshnizi A, Gao W, Udell M (2023) Optimus: Optimization modeling using mip solvers and large language models. *arXiv preprint arXiv:2310.06116* .

Bettencourt L (2024) Recent achievements and conceptual challenges for urban digital twins. *Nature Computational Science* 4(3):150–153.

Biggs M, Hariss R, Perakis G (2022) Constrained optimization of objective functions determined from random forests. *Production and Operations Management* 32(2):397–415, URL `http://dx.doi.org/10.1111/poms.13877`.

Chen Z, Xiong P (2023) Rsome in python: An open-source package for robust stochastic optimization made easy. informs journal on computing. *INFORMS Journal on Computing* 35(4):717–724.

China Academy of Urban Planning and Design (2023) 2023 china principal cities sharing bikes and sharing electric bikes riding report. `https://mp.weixin.qq.com/s/fLDNu3wg6Htg3RdIsBtybA`.

Chu Z, Chen J, Chen Q, Yu W, He T, Wang H, Peng W, Liu M, Qin B, Liu T (2023) A survey of chain of thought reasoning: Advances, frontiers and future. *arXiv preprint arXiv:2309.15402* .

Dong Q, Li L, Dai D, Zheng C, Wu Z, Chang B, Sun X, Xu J, Sui Z (2022) A survey for in-context learning. *arXiv preprint arXiv:2301.00234* .

Elmachtoub AN, Grigas P (2022) Smart "predict, then optimize". *Management Science* 68(1):9–26.

Gayialis SP, Kechagias EP, Konstantakopoulos GD (2022) A city logistics system for freight transportation: Integrating information technology and operational research. *Operational Research* 22(5):5953–5982.

Ge Y, Hua W, Mei K, Tan J, Xu S, Li Z, Zhang Y, et al. (2024) Openagi: When llm meets domain experts. *Advances in Neural Information Processing Systems* 36.

Jin Z, Wang Y, Lim Y, Pan K, Shen ZJM (2023) Vehicle rebalancing in a shared micromobility system with rider crowdsourcing. *Manufacturing & Service Operations Management* .

Kaggle (2020) Cycle share dataset. `https://www.kaggle.com/datasets/pronto/cycle-share-dataset`.

Kasneci E, Seßler K, Küchemann S, Bannert M, Dementieva D, Fischer F, Gasser U, Groh G, Günnemann S, Hüllermeier E, et al. (2023) Chatgpt for good? on opportunities and challenges of large language models for education. *Learning and individual differences* 103:102274.

Kojima T, Gu SS, Reid M, Matsuo Y, Iwasawa Y (2022) Large language models are zero-shot reasoners. *Advances in neural information processing systems* 35:22199–22213.

Li B, Mellou K, Zhang B, Pathuri J, Menache I (2023a) Large language models for supply chain optimization. *arXiv preprint arXiv:2307.03875* .

Li Q, Zhang L, Mak-Hau V (2023b) Synthesizing mixed-integer linear programming models from natural language descriptions. *arXiv preprint arXiv:2311.15271* .

Park JS, O'Brien J, Cai CJ, Morris MR, Liang P, Bernstein MS (2023) Generative agents: Interactive simulacra of human behavior. *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, 1–22.

Qi M, Shen ZJ (2022) Integrating prediction/estimation and optimization with applications in operations management. *Tutorials in Operations Research: Emerging and Impactful Topics in Operations*, 36–58 (INFORMS).

Qi M, Shi Y, Shi Y, Ma C, Yuan R, Wu D, Shen ZM (2023) A practical end-to-end inventory management model with deep learning. *Management Science* 69(2):759–773.

Qi W, Shen ZJM (2019) A smart-city scope of operations management. *Production and Operations Management* 28(2):393–406.

Wei J, Wang X, Schuurmans D, Bosma M, Xia F, Chi E, Le QV, Zhou D, et al. (2022) Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems* 35:24824–24837.

Wu Q, Bansal G, Zhang J, Wu Y, Zhang S, Zhu E, Li B, Jiang L, Zhang X, Wang C (2023) Autogen: Enabling next-gen llm applications via multi-agent conversation framework. *arXiv preprint arXiv:2308.08155* .

Yang C, Wang X, Lu Y, Liu H, Le QV, Zhou D, Chen X (2023) Large language models as optimizers. *arXiv preprint arXiv:2309.03409* .

Yao S, Yu D, Zhao J, Shafran I, Griffiths TL, Cao Y, Narasimhan K (2023a) Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601* .

Yao S, Zhao J, Yu D, Du N, Shafran I, Narasimhan K, Cao Y (2022) React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629* .

Yao W, Heinecke S, Niebles JC, Liu Z, Feng Y, Xue L, Murthy R, Chen Z, Zhang J, Arpit D, et al. (2023b) Retroformer: Retrospective large language agents with policy gradient optimization. *arXiv preprint arXiv:2308.02151* .

Zhang X, Yang Q (2023) Xuanyuan 2.0: A large chinese financial chat model with hundreds of billions parameters. *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, 4435–4439.

Zhang Y, Richter AR, Shanthikumar JG, Shen ZJM (2022a) Dynamic inventory relocation in disaster relief. *Production and Operations Management* 31(3):1052–1070.

Zhang Z, Zhang A, Li M, Smola A (2022b) Automatic chain of thought prompting in large language models. *arXiv preprint arXiv:2210.03493* .

Zhao A, Huang D, Xu Q, Lin M, Liu YJ, Huang G (2024) Expel: Llm agents are experiential learners. *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, 19632–19642.

A Notation

For the readers' convenience, we provide the notations used in City-LEO agent and E2E models as follows:

Table 4: Notations Table

Symbol	**Description**
Q	Natural language query proposed by users.
$f(\mathbf{x};\mathbf{w})$	QR-obj function.
$g(\mathbf{x};\mathbf{w})$	E2E programming.
$\bar{\mathbf{x}}$	Historical record of decisions.
$\bar{\mathbf{w}}$	Historical record of parameters.
$\mathcal{D}$	Empirical data $\mathcal{D} \stackrel{\text{def}}{=} (\bar{\mathbf{x}}, \bar{\mathbf{w}})$ recorded in Database.
P^{IG}	Prompts of QR-obj generator with Q&A instructions.
P^{PP}	Prompts of problem tailor with Q&A instructions.
$\mathbf{x}'$	Parameterized decisions.
$\hat{\mathbf{x}}$	Non-parameterized decisions.
S_t	Satisfaction factor in t-th iteration.
Sets	
$\mathcal{X}$	Feasible domain of decision variables.
$\mathcal{W}$	Parameters set.
I	Set of parking spots.
K	Set of discretized SOC levels of e-bike batteries.
Parameters	
$\mathbf{w}$	Parameters $\mathbf{w} \in \mathcal{W}$.
N	Capacity of the vehicle for dispatching e-bikes.
B	Capacity of the vehicle carrying fully-charged batteries.
C_i	Capacity of parking spot i.
v_{ik}	Initial number of e-bikes with SOC k at parking spot i.
r_{k_3}	Initial number of fully charged e-bikes on the dispatching vehicle.
ξ_i	Net traveling demand at parking spot i.
σ_i	Unmet traveling demand at parking spot i.
p_1	Penalty for using e-bikes with medium SOC level to satisfy demands.
p_2	Penalty for unmet demands.
Decision Variables	
$\mathbf{x}$	Decisions variables $\mathbf{x} \in \mathcal{X}$.
u_{ik}	Stock of e-bikes with SOC k at parking spot i after dispatching and battery swapping.
y_{ik}	Number of batteries with SOC k that are replaced by fully charged batteries at parking spot i.
z_{ik}	Number of e-bikes that are moved out or dropped off at parking spot i.

B Experiments Settings

B.1 Ground-truth Q&A

In this section, we provide the ground-truth Q&A instructions adopted in the prompt of City-LEO. Specifically, we provide accurate Q&A information (human-labeled) in three specific scenarios: *Operational*, *Customer-related*, and *Regulatory*. These scenarios encompass the majority of operational inquiries proposed by EBS operators. Within each scenario, there are typically 3 to 8 key QR-objs that are of utmost relevance to the majority of EBS operators (as shown in Table. 5). For each QR-obj, we provide a ground-truth "Answers" that is derived from human experience. This information is provided in the form of a Gurobi objective code.

B.2 Sample of Gurobi Code Created by City-LEO

This section presents the mathematical representation of the Gurobi code created by City-LEO, which is utilized to quantify the "Results Similarity" in the relevance test. The Gurobi code is shown below, which represents the objective of maximizing the total number of accessible e-bikes and its related mathematical essence:

Table 5: Ground-truth Objectives

Scenario	QR-obj	Ground-truth Obj-code
Operations	Proportion of bikes with lower SOC	`model.setObjective(gp.quicksum(model.getVarByName(f'u_{i}_{k0}') for i in range(55)), GRB.MINIMIZE)`
	Utilization of bike	`model.setObjective(gp.quicksum(model.getVarByName(f'u_{i}_{k}')for i in range(55) for k in [1, 2]), GRB.MAXIMIZE)`
	Charging cost	`model.setObjective(gp.quicksum(model.getVarByName(f'x_{i}_exchange_{k}') * (2 - k) for i in range(55) for k in [0, 1]), GRB.MINIMIZE)`
	Daily average battery exchange frequency	`model.setObjective(gp.quicksum(model.getVarByName(f'x_{i}_exchange_{k}') for i in range(55) for k in [0, 1]), GRB.MINIMIZE)`
	Market share	`model.setObjective(gp.quicksum(model.getVarByName(f'u_{i}_{k}') for i in range(55) for k in [1, 2]), GRB.MAXIMIZE)`
	Order cancellation rate	`model.setObjective(gp.quicksum(model.getVarByName(f'u_{i}_{k}') for i in range(55) for k in [1, 2]), GRB.MAXIMIZE)`
	Market penetration rate	`model.setObjective(gp.quicksum(model.getVarByName(f'u_{i}_{k}') for i in range(55) for k in [1, 2]), GRB.MAXIMIZE)`
Regulatory	overparking rate	`model.setObjective(gp.quicksum(gp.quicksum(model.getVarByName(f'u_{i}_{k}') for k in K) / C[i] for i in I) / len(I), GRB.MINIMIZE)`
	Carbon emission reduction	`model.setObjective(gp.quicksum((2 - k) * model.getVarByName(f'x_{i}_exchange_{k}')for i in range(55) for k in [0, 1]), GRB.MAXIMIZE)`
	Customer adoption	`model.setObjective(gp.quicksum(model.getVarByName(f'u_{i}_{k}') for i in range(55) for k in [1, 2]), GRB.MAXIMIZE)`
	Service coverage area	`model.setObjective(gp.quicksum(model.getVarByName(f'u_{i}_{k}') for i in range(55) for k in [1, 2]), GRB.MAXIMIZE)`
Customer	Complaint rate	`model.setObjective(gp.quicksum(k * gp.quicksum(model.getVarByName(f'u_{i}_{k}') for i in range(55)) for k in [1, 2]), GRB.MAXIMIZE)`
	Quality of service	`model.setObjective(gp.quicksum(model.getVarByName(f'u_{i}_{k}') for i in range(55) for k in [1, 2]), GRB.MAXIMIZE)`
	Customers' satisfaction level	`model.setObjective(gp.quicksum((k + 1) * model.getVarByName(f'u_{i}_{k}') for i in range(55) for k in [1, 2]), GRB.MAXIMIZE)`
	Accessible bike with higher SOC	`model.setObjective(gp.quicksum(model.getVarByName(f'u_{i}_{k}') for i in range(55) for k in [1, 2]), GRB.MAXIMIZE)`
	Average riding distance	`model.setObjective(gp.quicksum(model.getVarByName(f'u_{i}_{k}') * k for i in range(55) for k in [0, 1, 2]), GRB.MAXIMIZE)`
	Customers retention rate	`model.setObjective(gp.quicksum(model.getVarByName(f'u_{i}_{k}') for i in range(55) for k in [0, 1, 2]), GRB.MAXIMIZE)`
	Under growth rate	`model.setObjective(gp.quicksum(model.getVarByName(f'u_{i}_{k}') for i in range(55) for k in [1, 2]), GRB.MAXIMIZE)`
	Proportion of available bike	`model.setObjective((gp.quicksum(gp.quicksum(model.getVarByName(f'u_{i}_{k}') for k in K if k > k_0) / C[i] for i in I)), GRB.MAXIMIZE)`

- **Gurobi code**: `model.setObjective(gp.quicksum(model.getVarByName(f'u_{i}_{k}') for i in range(55) for k in [k_2, k_3]), GRB.MAXIMIZE)`
- **Mathematical essence**: $\max\ u_{0,1} + u_{0,2} + u_{1,1} + \underbrace{\cdots\cdots}_{u_{i,k}\ for\ i\ in\ [2,53]\ for\ k\ in\ [1,2]} + u_{54,1} + u_{54,2}$
- **Gurobi code**: `model.setObjective(gp.quicksum(gp.quicksum(model.getVarByName(f'u_{i}_{k}') for k in K if k >= k_2) for i in I), sense = GRB.MAXIMIZE)`
- **Mathematical essence**: $\max\ u_{0,1} + u_{0,2} + u_{1,1} + \underbrace{\cdots\cdots}_{u_{i,k}\ for\ i\ in\ [2,53]\ for\ k\ in\ [1,2]} + u_{54,1} + u_{54,2}$

The aforementioned two scenarios demonstrate that although City-LEO generates different Gurobi codes, the two codes nevertheless possess the same underlying mathematical essence.

B.3 Metrics of Accuracy Tests

We propose two criteria, *Global suboptimality* and *Local satisfaction gain*, to assess City-LEO's performance in terms of accuracy. Recall that FULL-obj refers to the initial objective function of the baseline model (i.e., FULL). Also, it represents the aggregate value of the leaf nodes in the random forest within the E2E model. The FULL-obj additionally represents the basis of EBS operations, with the goal of minimizing operating costs. Yet, achieving the second purpose (referred to as QR-obj) may conflict with the FULL-obj. Thus, we can adopt *global suboptimality* to analyze the compromising degree of FULL-obj when achieving users' QR-obj. Specifically, *global suboptimality* (g^{s}) can be formulated as follows:

$$g^{\mathrm{s}} \stackrel{\mathrm{def}}{=} \frac{\text{FULL-obj}_{\text{City-LEO}} - \text{FULL-obj}_{\text{FULL}}}{\text{FULL-obj}_{\text{FULL}}}$$

where $\text{FULL-obj}_{\text{City-LEO}}$ and $\text{FULL-obj}_{\text{FULL}}$ represent the optimal objectives of City-LEO and FULL in terms of initial objective function, respectively. A lower value of g^{s} suggests a lower level of compromise in attaining consumers' QR-obj with FULL-obj.

Secondly, the *local satisfaction gain* quantifies the "gains of QR-obj" achieved by the City-LEO model in terms of parking spots (refereed as the word "local") mentioned in user inquiries compared to the FULL. The FULL model does not involve any scoped-down operation. This measure (g^{l}) is formulated as follows:

$$g^{\mathrm{l}} \stackrel{\mathrm{def}}{=} \frac{\text{QR-obj}_{\text{City-LEO}} - \text{QR-obj}_{\text{FULL}}}{\text{QR-obj}_{\text{FULL}}}$$

where $\text{QR-obj}_{\text{City-LEO}}$ and $\text{QR-obj}_{\text{FULL}}$ represent the optimal values of QR-obj generated by City-LEO and FULL, respectively. These values are calculated based on the parking spots specified in user inquiries. Moreover, QR-obj's optimization direction (maximum or minimum) is frequently linked to the user's inquiry. For instance, increasing the number of available bikes while decreasing complaints represents a distinct path. In this regard, we separately computed the value of $\text{QR-obj}_{\text{City-LEO}} - \text{QR-obj}_{\text{FULL}}$ based on different QR-obj types. Therefore, if $g^{\mathrm{l}} \geq 0$, it means City-LEO outperforms FULL, and vice versa.

C Sample of Prompts

This section provides detailed information regarding prompt settings in the City-LEO agent architecture (See, Figure 2), which are categorized into Problem matcher, QR-obj generators, and Problem tailor.

C.1 Problem Matcher

The purpose of the Problem matcher is to match precise APIs that are encapsulated with corresponding QR-obj generators and E2E models. Therefore, the prompt only emphasizes instructing City-LEO to locate the most query-relevant APIs. The corresponding prompts are listed as follows:
Function description:

```
Extract the transportation domain and target from user's input.
```

Prompt template:

```
 User's questions are as follows:
{Input}
Based on the user's input, you need to extract the domain and the target that the user is
concerned about.  It is necessary to ensure that the output adheres to the prescribed
format.  Pay close attention to the details of the user's target, and extract the
information directly without summarizing it.
```

C.2 QR-obj Generator

The purpose of the QR-obj generators is to formulate the user's target into a objective function that is convex with decision variables and parameters, which are defined in the OR model. The formula is then translated into Gurobi syntax. The corresponding prompts are listed as follows:

Function description:

```
 Generate Python expressions that comply with solver syntax based on user's query.
```

Prompt template:

```
 You are a researcher and have a good knowledge of mathematics and operations research.
Your task is to generate a Python expression based on the given notations, ultimately
ensuring that the expression meets the syntax requirements of the solver (such as Gurobi)
and the output format meets specified standards.
For sets, parameters, and variables, the definitions are as follows:
{Notations}
Based on the above notations, you are required to provide a Python expression to depict
the target:
{Target}
The Python expression must be reasonable, containing a part of the mentioned notations
without adding other elements, and the formula must be convex.  Now I will provide you
with some ground-truth formulations for reference.
{Ground truth}
The output must contain the following points:
- First, provide explanation and illustrate why the formulation is set up in this way;
- Second, provide the corresponding Python code for the formula, and only include the
given notations;
- Third, rewrite Python code to conform to solver syntax.
It is necessary to ensure that the output is in the specified format.  Take a deep breath
and solve the problem step by step.  Now you can start working.
```

C.3 Problem Tailor

The purpose of the Problem tailor is to identify and exclude parking spots from database tables that are not related to the user's concerns. Furthermore, the Problem tailor provides reasons for selected spots and include contextual information about them. The corresponding prompts are listed as follows:

Function description:

```
 Selecting low correlation stations parameterization based on SQL.
```

Prompt template:

```
 You are a database administrator and an expert in operations research.  You need to
perform cross-analysis on three existing tables in the database and ultimately provide me
with some station names and why you selected these station.
Explanation of the data tables as follows:
{Database description}
Based on the database information provided, you need to tell me which stations have
little impact on stations within the target:
{Target}
That means it's unnecessary to make decisions, and then all decision variables involved
can be parameterized from decisions to parameters.  The current Python expression of the
target is:
```

{Formulation}
For sets, parameters, and variables, the definitions are as follows:
{Notations}
You should pay attention to the following points:
- First, you must select at least *{Min}* stations and a maximum of *{Max}* stations. That means no more than the limit and no less than the requirement. Anyway, you need to give me an answer and you must not choose the stations involved in user targets.
- Second, provide me with all the stations to be parameterized for subsequent conversion into parameters.
- Third, Explain the reason for your choice.
It is necessary to ensure that the output is in the specified format. Take a deep breath and solve the problem step by step. Now you can start working.

C.4 Code Safeguard

The purpose of the Code safeguard is to go over the erroneous Python code or invalid Gurobi syntax generated by previous steps. After LLM provides an incorrect formulation, Code safeguard will debug each error until the code is able to execute without any issues. The corresponding prompts are listed as follows:
Function description:

Fix errors in the code without changing semantics to ensure it runs properly

Prompt template:

You are a programmer specializing in operations research, meticulous and proficient in Gurobi and Python syntax. You are now required to correct the erroneous code. The current Gurobi code in Python is:
{Gurobi code}
But this code has some errors, and the detailed error information are as follows:
{Error traceback}
You need to modify this code to address the errors specifically, ensuring it can run properly without altering its overall structure. The output must comply with the format requirements.
Now I will give you some ground truth formulation. They can be run correctly. If the ground truth I provide to you is valuable, you can refer to it.
{Ground truth}
For sets, parameters, and variables, the definitions are as follows:
{Notations}
It is necessary to ensure that the output is in the specified format. Take a deep breath and solve the problem step by step. Now you can start working.

C.5 Sample Safeguard

The purpose of the Sample safeguard is to assess the rationality and compliance of the selected parking spots. After the Problem tailor outputs certain spots, Sample safeguard will reevaluate them and make adjustments if they are deemed inappropriate. The corresponding prompts are listed as follows:
Function description:

Check the robustness of the selection to ensure that the results are compliant and reliable.

Prompt template:

The selection content given by the previous work is as follows:
{Selection}
The selection content is based on a database search to select station for parameterization.
You should pay attention to the following points:
- Ensure that your output meets the output format specifications.
- Verify the reliability of the current selection. The station must be within the given range and the reason for choosing should be sufficient.

```
Verify the above two points, and if they do not meet the requirements, make corrections
to your results until they fulfill the request.  Now, I'll give you some references to
support you in completing tasks.
{Notations}
{Database description}
Take a deep breath and solve the problem step by step.  Now you can start working.
```

D Performance of the Scope-down Policy

We conduct additional experiments to examine how the City-LEO scopes down the EBS parking spots that are relevant to the user's query. More precisely, we set the "maximum allowable number of parameterized parking spots" option under the same settings in the aforementioned experiments. Figure 8 illustrates the number of parameterized parking spots as the maximum allowed number of parameterized parking spots increases. The figure plots both the mean and median numbers of scoped-down parking spots, depicted by a red line for the mean and a black line for the median across different settings of parameterized parking spots (10, 25, 40, 55). The box plots provide an overview of the distribution of scoped-down parking spots at each parameter setting.

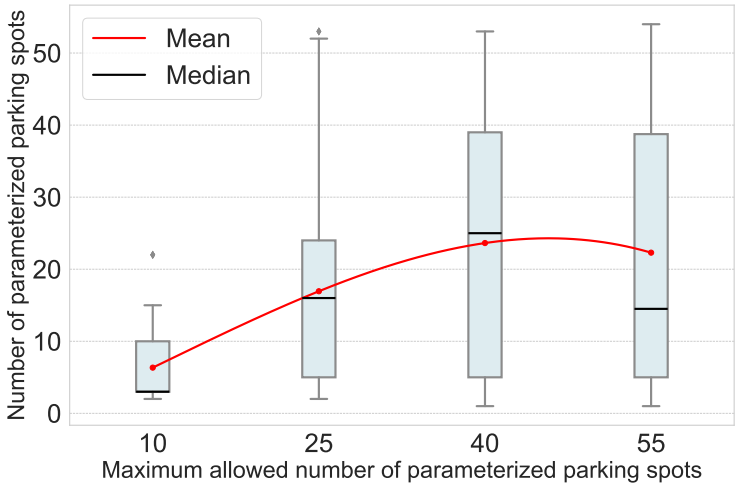


Figure 8: Average Scoped-down Parking Spots Evaluation

From the Figure 8, it can be observed that as the maximum allowed number of parameterized parking spots increases, there is a general trend of increasing mean and median numbers of parameterized parking spots. Specifically, when the maximum allowed parking spots are set at 10, the median number of scoped-down parking spots is relatively low, around 10. This increases to approximately 20, 30, and 40 as the allowed parameterized parking spots are increased to 25, 40, and 55, respectively.

The red line indicating the mean number of scoped-down parking spots shows a steady rise, reinforcing the trend observed in the medians. The spread of the data, as indicated by the whiskers of the box plots, suggests increasing variability in the number of scoped-down parking spots as more spots are allowed to be parameterized, particularly noticeable at the higher settings of 40 and 55.