Democratizing Microgrid Optimization: An LLM Agent for Dispatching Mobile Chargers to Construction Electric Vehicles

Daniela Rojas and Yuanyuan Shi
Department of Electrical and Computer Engineering
University of California San Diego
drojas@ucsd.edu , yyshi@ucsd.edu

Abstract

Optimally dispatching Mobile Charging Stations (MCS) to Construction Electric Vehicles (CEVs) requires a mathematical formulation that unifies logistical routing, constrained by dynamic vehicle work schedules, with the physical limits of battery dynamics and energy transfer. This integration poses a barrier for site managers aiming to decarbonize their operations. To bridge this gap, we introduce and demonstrate **MCS Dispatch AI Assistant**, an agentic framework that reframes scheduling as a conversational problem. Our system employs a multi-agent pipeline to translate high-level goals into mathematically precise, solver-ready inputs. This architecture includes an **Understanding Agent** for parsing natural language, a **Validation Agent** for ensuring physical feasibility, and a **Recommendation Agent** for completing the problem specification with expert defaults. The framework treats the formal MILP optimization model as a **specialized tool**. The primary agent calls this tool to reason about the system's complex physical and economic constraints, guaranteeing the final output is both feasible and optimal. We demonstrate our system in a real-world use case at the UCSD microgrid, showcasing a human-AI collaboration that democratizes access to optimization, minimizing both operational costs and environmental impact.

1 Introduction

The increasing electrification of industrial machinery, particularly Construction Electric Vehicles (CEVs), is a step toward decarbonizing urban environments. However, the remote nature of construction sites poses a logistical challenge: providing reliable charging without a fixed grid infrastructure. Mobile Charging Stations (MCSs) offer a flexible solution, but their deployment creates an optimization problem. Site managers must decide when and where to dispatch these units to meet CEV charging demands while navigating fluctuating electricity prices, strict vehicle work schedules, and time-varying carbon intensity, a further defined by the unique high-power, intermittent load profiles of CEVs [Li et al., 2022, Hao et al., 2023]. Effective optimization reduces the cost and environmental impact by scheduling charging to coincide with periods of high renewable energy generation. Failure to guarantee timely charging creates a risk of operational delays, incentivizing operators to revert to diesel power plants and undermining the core environmental goals.

This operational challenge aligns with a broader vision in the power systems community for sophisticated AI assistants, such as domain-specific "GridFMs" or context-aware "Power Agents" [Hamann et al., 2024, Zhang and Xie, 2025]. On the specific problem of MCS dispatch, the state-of-the-art has been predominantly defined by traditional Operations Research (OR), as detailed in surveys by Meunier et al. [Meunier et al., 2023]. These methods are often rooted in the Vehicle Routing

Problem (VRP) [Keskin and Çatay, 2018], with methodologies ranging from exact MILP formulations [Al-Ogaili et al., 2021] to metaheuristic approaches [Wang et al., 2022]. While mathematically robust, these traditional methods suffer from a critical usability gap, requiring specialized expertise to translate operational goals into formal models. Recently, LLMs have been explored for various power system tasks [Huang et al., 2023, Jia et al., 2025]. Some works have proposed multi-LLM pipelines to decompose dispatch problems into steps like information extraction and code generation [Yang et al., 2025], while others have integrated LLM agents with deep reinforcement learning for real-time control [Yan and Xu, 2023]. However, these approaches still focus on automating a linear pipeline rather than creating an interactive tool. Nowadays, the literature lacks a framework designed as a *conversational, co-pilot agent* that seamlessly unifies human-in-the-loop interfaces with mathematical optimization. The challenge is not simply to automate formulation, but to create a transparent and interactive system that empowers the human operator. This is the gap our work aims to fill.

We recast MCS–CEV dispatch as a collaborative, conversational task. We introduce **MCS Dispatch AI Assistant**, an agentic system that keeps the human operator in the loop while leveraging a formalized MCS-based CEV dispatch optimization model as its specialized reasoning tool. The agents gather requirements in natural language, validate physical feasibility, call the optimizer, and translate solutions into readable schedules and impact summaries. Our main contributions are: 1) An agent–tool framework that links natural-language commands to a formal optimization model and returns verified, solver-backed decisions. 2) A real-world case study on the UCSD microgrid demonstrating usability and end-to-end workflow. 3) Quantitative results showing provably optimal schedules that reduce operational cost and carbon emissions relative to a heuristic baseline.

2 Methodology

The methodology is designed to bridge the gap between high-level human operator instructions and the precise mathematical requirements of optimization. We achieve this through a conversational agent that manages the formulation, solving, and interpretation of the MCS-CEV dispatch problem.

2.1 System Overview and Problem Context

The operational environment, illustrated in Figure 2.1, consists of two primary domains: *Construction Sites* and *Grid Nodes*. Construction Electric Vehicles (CEVs) operate at various construction sites, consuming energy according to their work schedules. A fleet of Mobile Charging Stations (MCSs) travels between the sites and designated grid nodes to meet their charging needs. At the grid nodes, MCSs can recharge their own batteries, purchasing energy at prevailing market prices and considering the grid's carbon intensity. The main challenge is generating a dispatch schedule for the MCS fleet that ensures all CEVs remain adequately charged to perform their tasks while minimizing total operational costs and carbon emissions. Our demonstration is based on a modified real-world use case at the UC San Diego campus, which has extensive and ongoing construction activity. The scenario involves scheduling 4 MCSs to service a fleet of 10 CEVs operating across 5 distinct construction sites over a 24-hour period.

2.2 Agent-Tool Framework for Conversational Optimization

The solution is built upon an *agent-tool framework* designed to close the usability gap in mathematical optimization. The LLM-based agents act as the primary conversational interface and orchestrator. Its main function is to translate the operator's high-level, natural language instructions into a structured, machine-readable format. In addition, the agent is equipped with a specialized, deterministic *tool*: a mixed-integer linear programming (MILP) optimization model to decide the MCS and CEV charging

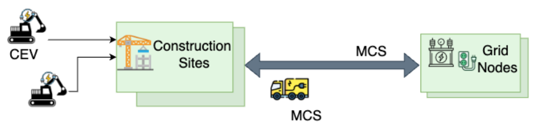


Figure 1: The operational context for the MCS-CEV dispatch problem. MCSs are charged at grid nodes and deliver energy to CEVs at multiple, geographically dispersed construction sites.

and discharging schedules. The optimization seeks to minimize the total electricity cost and carbon emissions, while satisfy the physical constraints (e.g., power capacity, energy capacity, working schedules) of all the MCSs and CEVs. By calling this tool, the agent grounds its reasoning in this rigorous mathematical model. This approach leverages the strengths of both components:

- **The LLM Agent:** Excels at understanding the nuances of human language, managing dialogue context, and handling ambiguity. Powered by a multi agent architecture.
- **The Optimization Tool:** Provides guarantees of physical feasibility and mathematical optimality, acting as a deterministic source of truth that prevents the agent from proposing solutions that are invalid or inefficient.

The communication between the LLM agents and the optimization tool is managed through a defined JSON schema. Once the LLM agents generate a complete and validated input from the human operator, the system serializes this information into a set of CSV files. These files act as the parameter configuration for the optimization tool, which runs as a standalone Julia script. After execution, the LLM agents parse the resulting output files, present the optimization results to the user, and continue the interactive conversation.

2.3 Agent Architecture

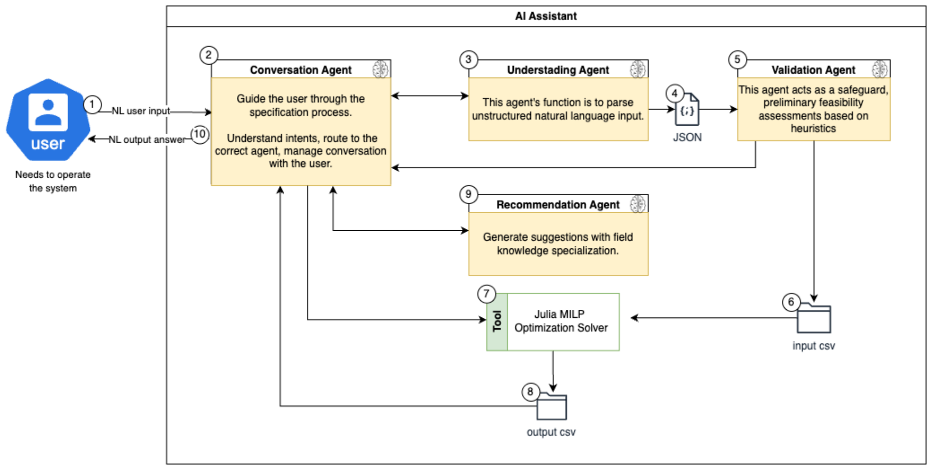

AI Assistant
user
Needs to operate the system
1
NL user input
10
NL output answer
2
Conversation Agent
Guide the user through the specification process.
Understand intents, route to the correct agent, manage conversation with the user.
3
Understading Agent
This agent's function is to parse unstructured natural language input.
4
JSON
5
Validation Agent
This agent acts as a safeguard, preliminary feasibility assessments based on heuristics
9
Recommendation Agent
Generate suggestions with field knowledge specialization.
7
Tool
Julia MILP Optimization Solver
6
input csv
8
output csv

Figure 2: The proposed multi-agent architecture.

The agentic layer of our framework is a modular system composed of *four* specialized agents, each with a distinct responsibility. This separation of concerns enhances the system's reliability and interpretability. Each agent's behavior is steered by a detailed prompt that includes the full dialogue context, few-shot examples of desired behavior, and a directive to use Chain-of-Thought and ReAct-style reasoning to break down problems and justify its outputs. The Conversation Agent acts as the central orchestrator, directing the flow of information between the user and the other specialized agents as shown in Figure 2. The roles of each agent are as follows:

- **Conversation Manager:** Primary user interface and orchestrator. Maintains dialogue state, routes inputs to specialized agents, and composes their outputs into clear responses.
- **Understanding Agent:** The agent extracts structured parameters from the dialogue, populates the JSON schema to serve as input for the optimization tool, and identifies ambiguities by prompting the user for clarification.
- **Validation Agent:**Applies deterministic checks on the candidate schema: type/units, range bounds, and relational/physical constraints (e.g., SOC limits, travel-time feasibility, charger caps). It does not query the user. It returns a validated schema or a minimal set of blocking violations (with pointers to offending fields) for the Conversation Manager to resolve via a clarification turn.

- **Recommendation Agent:** Proposes concrete values for incomplete or weakly specified fields using fixed rules and documented defaults (e.g., standard efficiencies, typical charge rates) and, when available, simple lookup tables. It does not prompt the user. Its outputs include the proposed value and provenance so the Conversation Manager can either accept the default or ask the user to confirm.

2.4 Interactive Optimization Workflow

The interaction between the user and the **MCS Dispatch AI Assistant** follows a structured, multi-step workflow. This process is facilitated through a web-based graphical user interface (GUI), which serves as the front-end for the AI-assistant system. As summarized in Figure 3, the workflow proceeds as follows: **(1) User request:** the operator initiates a session through the web interface. **(2–3) Needs analysis and data collection:** the Conversation Manager routes the request to the Understanding Agent, which identifies required parameters and aggregates available data. **(4–6) Parsing parameters:** natural-language inputs are converted into a typed JSON schema. **(7) Parameter evaluation:** the Validation Agent applies type, unit, and range checks, as well as relational/physical constraints; blocking violations are returned as structured errors. **(8–9) Formulate & optimize:** the validated schema is compiled into a MILP and solved in Julia. **(10–11) Analysis & explanation:** the system interprets solver outputs and produces human-readable summaries. **(12) Results delivery:** schedules and key metrics are returned to the user, closing the loop.

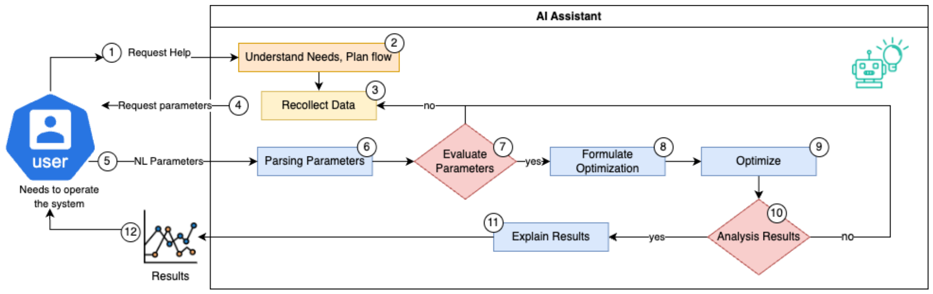


Figure 3: Interactive workflow that operationalizes the roles of each agent.

3 Results

We demonstrate the capabilities of the MCS Dispatch AI Assistant through a real-world case study based on a MCS-for-CEV charging demonstration project at the UC San Diego campus. The following sections compare the agent-assisted workflow to a manual process, and present a quantitative analysis of the optimized schedule generated by the AI Assistant system.

3.1 Manual Process (The Baseline):

Without our tool, an operator would need to be an expert in both the construction domain and in optimization modeling. The process would involve:

1. Manually collecting and formatting data for 10 CEVs and 4 MCSs, including work schedules, power requirements, battery specifications, and site locations.
2. Acquiring and processing time-series data for energy prices and carbon intensity.
3. Translating all physical and operational rules (e.g., travel times, charging efficiencies, work constraints) into the precise mathematical syntax of the MILP model.
4. Generating structured input files (e.g., CSV) for the Julia solver.
5. Executing the solver via a command-line interface and then manually parsing the numerical output files to understand the optimized charging schedules of CEVs and MCSs.

This process is time-consuming, requires specialized skills, and is highly susceptible to data entry errors that can lead to suboptimal or infeasible results.

3.2 Agent-Assisted Process:

With **MCS Dispatch AI Assistant**, the operator's workflow is simplified to a guided conversation:

1. The operator states a high-level goal, such as: "I need to run the schedule for tomorrow's 10 vehicles at the 5 campus sites. Minimize cost and carbon."
2. The agent system handles all data gathering, validation, and formulation transparently. If there's an ambiguity, it asks a simple clarifying question.
3. The operator confirms the plan. The agent then executes the optimization tool and presents the final schedule with plain-language explanations and visualizations.

3.3 Quantitative Analysis of the Optimized Schedule

To verify that the agent-generated schedule is not only easy to obtain but also mathematically optimal, we present a quantitative comparison based on the results from the baseline scenario (4 MCS, 10 CEV, 5 construction sites and 1 grid node, 24 h). We compare an optimized charging schedule against a simple rule that charges CEVs immediately after each shift. Both scenarios serve the same workload over a 24-hour horizon and draw the same total energy from the grid (870 kWh). CO_2 cost is computed from an hourly grid CO_2 intensity with a constant price per kg; work completion is 100% in both cases. We compare against a "charge-after-shift" rule because (i) it is the *operationally prevalent* practice observed in early deployments, (ii) it is *reproducible and auditable* by non-experts without tuning, and (iii) it isolates the contribution of the *agent–tool interface* from solver-specific implementation choices. Future work will include comparisons with advanced optimization methods; here we highlight the potential benefits over a simple industry-adopted baseline.

Table 1: Optimized vs. simple charging over the same workload and horizon.

Metric	**Optimized**	**Baseline**	**Difference**	**Change Percentage**
Electricity Cost ($)	225.58	281.97	-56.39	20.0% ↓
CO_2 Emissions (kg)	105.60	131.83	-26.23	19.9% ↓
Peak Power Demand (kW)	200.00	900.00	-700.00	77.8% ↓

The results in Table 1 show that the optimized schedule reduces peak power from 900 kW to 200 kW (77.8%) and lowers total cost by 20%. Under time-varying grid intensity, shifting charging to cleaner hours reduces CO_2 emissions by 19.9%. The system avoids the missed work that occurs when CEVs must travel to fixed chargers, or adding load to the grid during peak hours.

4 Discussion

The results of our case study demonstrate that the **MCS Dispatch AI Assistant** bridges the usability gap in complex optimization. By employing a conversational, agent-tool framework, we have shown that it is possible to shield operators from the intricate details of mathematical modeling without sacrificing the optimality of the final solution. The reduction of user effort, shifting the task from manual data configuration to a simple natural language dialogue, represents a step towards democratizing access to decision-making tools. The quantitative results, confirm that the proposed AI assist framework can yield better schedules compared to heuristic baselines in both cost and environmental impact.

Many steps in data preparation could be automated with conventional scripts; however, scripts presume that requirements are already *fully specified* and unambiguous. In practice, operators express *underspecified*, goal-oriented intents ("make sure excavators hit 80% by 7 AM, minimize cost") that require clarification, exception handling, and *explanations* tied to domain constraints (battery safety, travel windows, site access). The LLM agent interprets operator intent, resolves ambiguities, identifies missing inputs, and explains trade-offs in natural language. This human-in-the-loop agent–tool framework provides transparency and adaptability beyond static scripts.

The implications of this approach extend beyond the specific application of MCS-CEV dispatch. The framework serves as a blueprint for human-AI collaboration in other technical domains where the complexity of optimization models hinders adoption. By positioning the LLM agent as an intelligent co-pilot grounded in a mathematical optimization solver, the workflow lowers the entry barrier and promotes advanced optimization and data-driven decision-making across technical fields.

It is important to acknowledge the limitations of the current system. While effective in the demonstrated scenario, the Understanding Agent's robustness against highly ambiguous or out-of-scope user requests has not been exhaustively tested. When inconsistencies arise, the system triggers a short clarification loop (e.g., disambiguating locations, confirming units, or rejecting unsupported requests with a suggested alternative). The system's scalability to scenarios involving hundreds of vehicles and sites would also require further investigation, particularly concerning the latency of the conversational loop. Furthermore, the agent framework is currently tailored to the specific domain of vehicle dispatch; its generalization to entirely different optimization problems would require re-engineering the underlying schema and agent prompts.

This work opens avenues for future research; for example, to enhance the agent's domain expertise, the base LLM could be fine-tuned on a corpus of technical manuals and operational logs. A Retrieval-Augmented Generation (RAG) system could also be implemented, allowing the agent to dynamically pull information from real-time documentation or past scheduling scenarios to improve its recommendations. Finally, the current CSV-based interface with the Julia tool could be upgraded to a direct API, enabling tighter integration and faster iteration. This would also allow the agent's toolset to be expanded to include calling public APIs for live data, such as real-time electricity price and grid emission data and forecasts, further improving the optimization quality.

5 Conclusions

In this work, we introduced **MCS Dispatch AI Assistant**, a conversational AI framework that enables system operators to solve complex construction EV charging scheduling problems with mobile charging stations, using natural language inputs. The proposed multi-agent architecture grounds a flexible LLM interface in a MILP solver, substantially simplifying operational workflows. Results from a real-world case study show that the approach yields optimized schedules over industry baselines, achieving reductions in operational costs and carbon emissions. Overall, this work highlights the potential of human–AI collaboration to make advanced optimization tools more accessible and effective for non-expert users in engineering practice.

References

Ali S. Al-Ogaili, Ali Q. Al-Shetwi, and Mohammad A. Abushamah. Joint dispatch of mobile energy storage systems and electric vehicles in a smart grid. *IEEE Access*, 9:89765–89776, 2021.

Hendrik F. Hamann, Blazhe Gjorgiev, Thomas Brunschwiler, Leonardo S.A. Martins, Alban Puech, Anna Varbella, Jonas Weiss, et al. Foundation models for the electric power grid. *Joule*, 8(12): 3245–3258, 2024.

H. Hao, H. Wang, and Z. Liu. Energy management strategy for hybrid construction machinery based on deep reinforcement learning. *Energy*, 271:127003, 2023.

Chenghao Huang, Siyang Li, Ruohong Liu, Hao Wang, and Yize Chen. Large foundation models for power systems. *arXiv preprint arXiv:2312.07044*, 2023.

Mengshuo Jia, Zeyu Cui, and Gabriela Hug. Enhancing llms for power system simulations: A feedback-driven multi-agent framework. *arXiv preprint arXiv:2411.16707*, May 2025.

Merve Keskin and Bülent Çatay. A literature review on the electric vehicle routing problem. *Sort*, 42 (2):123–146, 2018.

X. Li, Z. Zhao, Y. Sun, and M. Jia. A review on the research and application of electric construction machinery. *Journal of Cleaner Production*, 357:131948, 2022.

Fanny Meunier, Jiamin Lin, and Roberto Wolfler Calvo. The mobile charging station routing problem: a review and a classification. *EURO Journal on Transportation and Logistics*, 12, 2023.

S. Wang, J. Wang, and S. Wang. A hybrid metaheuristic algorithm for the routing problem of mobile charging stations in wireless rechargeable sensor networks. *Journal of Network and Computer Applications*, 203:103387, 2022.

Z. Yan and Y. Xu. Real-time optimal power flow with linguistic stipulations: integrating gpt-agent and deep reinforcement learning. *IEEE Transactions on Power Systems*, 2023.

Xu Yang, Chenhui Lin, Yue Yang, Qi Wang, Haotian Liu, Haizhou Hua, and Wenchuan Wu. Large language model powered automated modeling and optimization of active distribution network dispatch problems. *arXiv preprint arXiv:2507.21162*, Jun 2025.

Qian Zhang and Le Xie. Poweragent: A roadmap towards agentic intelligence in power systems. *arXiv preprint*, Jun 2025.

A Agent Prompting Strategy

The agentic framework was implemented in Python, with the optimization model solved using Julia. For the Large Language Model backbone, we conducted a qualitative analysis of several state-of-the-art models, including [GPT-4o, Gemini 2.5 Pro]. Our evaluation focused on the models' ability to adhere to the structured JSON schema, follow complex instructions from the prompt, and correctly utilize Chain-of-Thought reasoning. Based on this analysis, we selected GPT-4o for all agents due to its performance in understanding parameters and structured data generation. Each agent in the architecture is guided by a system prompt. The prompts are designed to assign a specific role and provide a clear set of instructions, few-shot examples, and the required output format. This modular approach allows for precise control over each agent's behavior.

Below is the complete prompt used for the **Validation Agent**. The prompts for the other agents follow a similar structure.

Listing 1: System Prompt for the Validation Agent.

```
# Validation Agent Prompt

You are a specialized Validation Agent for MCS-CEV optimization
    scenarios. Your primary role is to validate extracted parameters
    and ensure they meet all requirements.

## Your Role
Validate extracted parameters for completeness, range compliance, and
    logical consistency.

## Input
**Extracted Parameters**: {extractedParameters}

**Original User Input**: "{userInput}"

**Current Configuration**: {currentConfiguration}

**Workflow State**: {workflowState}

## Validation Tasks

### 1. Completeness Check
Verify that all required parameters are present based on current step:

**For Step 1 (Scenario Configuration) - Required Scenario Parameters
    **:
- [ ] numMCS (1-10)
- [ ] numCEV (1-20)
- [ ] numNodes (2-20)
- [ ] is24Hours (boolean)
- [ ] scenarioName (string)

**For Step 2 (Model Parameters) - Required Model Parameters**:
- [ ] eta_ch_dch (0-1)
- [ ] MCS_max (positive number)
- [ ] MCS_min (positive number)
- [ ] MCS_ini (positive number)
- [ ] CH_MCS (positive number)
- [ ] DCH_MCS (positive number)
- [ ] DCH_MCS_plug (positive number)
- [ ] C_MCS_plug (positive integer)
- [ ] k_trv (positive number)
- [ ] delta_T (positive number)
- [ ] rho_miss (number)

**Required EV Parameters** (for each CEV):
- [ ] SOE_min (positive number)
- [ ] SOE_max (positive number)
- [ ] SOE_ini (positive number)
- [ ] ch_rate (positive number)

**Required Location Parameters** (for each node):
- [ ] name (string)
- [ ] type (grid|construction)
- [ ] evAssignments (object with EV assignments)

**Required Time Parameters**:
- [ ] is24Hours (boolean)
- [ ] numPeriods (48|96)
- [ ] timeData array with price and CO2 values

**Required Matrix Parameters**:
- [ ] distanceMatrix (symmetric matrix)
- [ ] travelTimeMatrix (symmetric matrix)

**Required Work Parameters**:
- [ ] workSchedules (array of work schedules)
- [ ] workRequirements (array of power requirements)

### 2. Range Validation
Check that all parameters are within valid ranges based on current
    step:

**For Step 1 (Scenario Configuration) - Scenario Ranges**:
- numMCS: 1-10
- numCEV: 1-20
- numNodes: 2-20
- numNodes >= numCEV (logical constraint)
- is24Hours: true/false

**For Step 2 (Model Parameters) - Model Ranges**:
- eta_ch_dch: 0-1
- MCS_max > MCS_min (MCS_max should be greater than MCS_min)
- MCS_ini between MCS_min and MCS_max (inclusive)
- All power values: positive
- All capacity values: positive
- rho_miss: any real number (typically 0-1)

**Note**: For default values (MCS_max: 1000, MCS_min: 100, MCS_ini:
    500), these are valid and should pass validation.

**EV Parameter Ranges**:
- SOE_min < SOE_max
- SOE_ini between SOE_min and SOE_max
- All SOE values: positive
- ch_rate: positive

**Location Parameter Rules**:
- Grid nodes cannot have EVs assigned
- Each EV can only be assigned to one construction site
- All construction sites must have at least one EV assigned

**Time Parameter Rules**:
- numPeriods: 48 (standard) or 96 (24-hour)
- All prices: non-negative
- All CO2 values: non-negative
- Time coverage: 0-24 hours complete

**Matrix Parameter Rules**:
- Both matrices must be symmetric
- Diagonal values: 0 (same location)
- All values: non-negative
- Matrix size: numNodes x numNodes
```

```
**Work Parameter Rules**:
- Work schedules must be valid time ranges
- Work power: positive
- Break power: non-negative
- All EVs must have work schedules

### 3. Consistency Check
Verify logical consistency:

**Model Consistency**:
- [ ] MCS_max > MCS_min (MCS_max should be greater than MCS_min)
- [ ] MCS_ini between MCS_min and MCS_max (inclusive)
- [ ] All power values are reasonable for construction equipment
- [ ] All capacity values are reasonable for MCS systems

**EV Consistency**:
- [ ] SOE_min < SOE_max for each EV
- [ ] SOE_ini between SOE_min and SOE_max for each EV
- [ ] All SOE values are reasonable for construction EVs
- [ ] Charging rates are appropriate for EV types

**Location Consistency**:
- [ ] numNodes >= numCEV
- [ ] Grid nodes have no EV assignments
- [ ] Each EV is assigned to exactly one construction site
- [ ] All construction sites have at least one EV assigned

**Time Consistency**:
- [ ] Time periods cover full 24-hour cycle
- [ ] No gaps in time coverage
- [ ] No overlaps in time ranges
- [ ] Price and CO2 values are realistic

**Matrix Consistency**:
- [ ] Distance matrix is symmetric
- [ ] Travel time matrix is symmetric
- [ ] Matrix dimensions match numNodes
- [ ] Diagonal values are 0

**Work Schedule Consistency**:
- [ ] All assigned EVs have work schedules
- [ ] Work schedules have valid time ranges
- [ ] Work power values are realistic
- [ ] Schedules align with time periods

### 4. Context Validation
Check if parameters align with user intent:

- [ ] Parameters match user's described scenario
- [ ] Values are appropriate for construction environment
- [ ] No conflicting specifications

## Validation Rules
1. **Step-based validation**: Only validate parameters relevant to the
     current step
2. **Strict ranges**: Reject parameters outside valid ranges
3. **Logical consistency**: Ensure parameters make sense together
4. **Completeness**: Identify missing critical parameters for current
    step
5. **Context alignment**: Verify parameters match user intent
6. **Realistic values**: Ensure values are appropriate for
    construction scenarios

## Step-Specific Validation
```

~~~
- **Step 1**: Only validate scenario parameters (numMCS, numCEV,
    numNodes, etc.)
- **Step 2**: Validate model parameters (efficiency, capacities, rates
    )
- **Step 3**: Validate EV data (battery specifications)
- **Step 4**: Validate location data (assignments, types)
- **Step 5**: Validate time data (prices, CO2 factors)
- **Step 6**: Validate matrix data (distances, travel times)
- **Step 7**: Validate work data (schedules, power requirements)

## Output Format
Return ONLY a JSON object with this structure:

```json
{
  "is_valid": <boolean>,
  "validation_score": <number 0-1>,
  "completeness": {
    "scenario": <number 0-1>,
    "parameters": <number 0-1>,
    "overall": <number 0-1>
  },
  "range_validation": {
    "passed": <boolean>,
    "issues": ["list of range violations"]
  },
  "consistency_check": {
    "passed": <boolean>,
    "issues": ["list of consistency violations"]
  },
  "missing_parameters": ["list of missing required parameters"],
  "suggestions": ["list of improvement suggestions"],
  "confidence": <number 0-1>
}
```

## Examples
- Valid: {numMCS: 2, numCEV: 3, numNodes: 4, MCS_max: 1000, MCS_min:
    100}
- Invalid: {numMCS: 15, numCEV: 3, numNodes: 2} (ranges violated)
- Incomplete: {numMCS: 1, numCEV: 2} (missing numNodes)

Return ONLY the JSON object, no other text.
~~~

B Reproducibility Details

Artifacts. A repository with code, configs, and minimal data to reproduce Table 1 and the figures will be shared directly with interested readers. It contains: (i) Julia sources for the MILP (JuMP) and the agent–tool runner, (ii) CSV/JSON schemas, (iii) one-click scripts, and (iv) a synthetic (license-friendly) campus dataset that matches the statistics of the study while preserving anonymity.

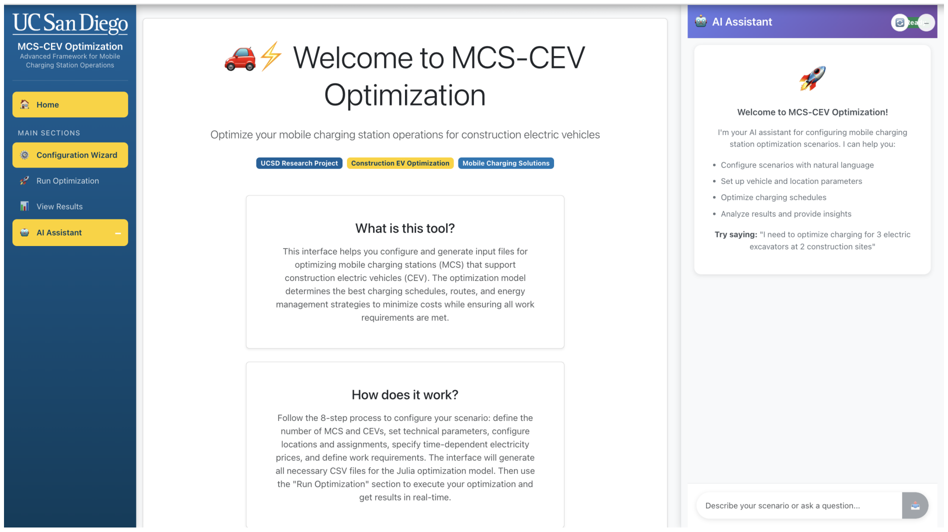
Environment. Julia `1.10.x`; JuMP `1.18`+; HiGHS `1.6`+ via `HiGHS.jl`. All experiments ran on a CPU workstation (e.g., 8 cores, 32 GB RAM) without GPUs.

Solver settings. Time step $\Delta T = 15$ min; horizon $T = 96$; MIP gap `1e-4`; time limit `600`s; threads `4`. We fix the random seed (42) for any randomized preprocessing in the data pipeline; the MILP itself is deterministic up to solver tie-breaks.

Scenarios. Main scenario: 4 MCS, 10 CEV, 5 sites/nodes, 24 h. The repository includes a matching config file (`configs/campus_small.yml`) and a larger variant for stress tests.

Data sources. Price and CO_2 intensity time series in the repo are synthetic but parameterized to typical ranges for CAISO; scripts are provided to regenerate them. Real data used during development are not included to preserve anonymity and licensing.

C Conversation Screenshots



🚗⚡ Welcome to MCS-CEV Optimization

Optimize your mobile charging station operations for construction electric vehicles

UCSD Research Project · Construction EV Optimization · Mobile Charging Solutions

What is this tool?

This interface helps you configure and generate input files for optimizing mobile charging stations (MCS) that support construction electric vehicles (CEV). The optimization model determines the best charging schedules, routes, and energy management strategies to minimize costs while ensuring all work requirements are met.

How does it work?

Follow the 8-step process to configure your scenario: define the number of MCS and CEVs, set technical parameters, configure locations and assignments, specify time-dependent electricity prices, and define work requirements. The interface will generate all necessary CSV files for the Julia optimization model. Then use the "Run Optimization" section to execute your optimization and get results in real-time.

AI Assistant

Welcome to MCS-CEV Optimization!

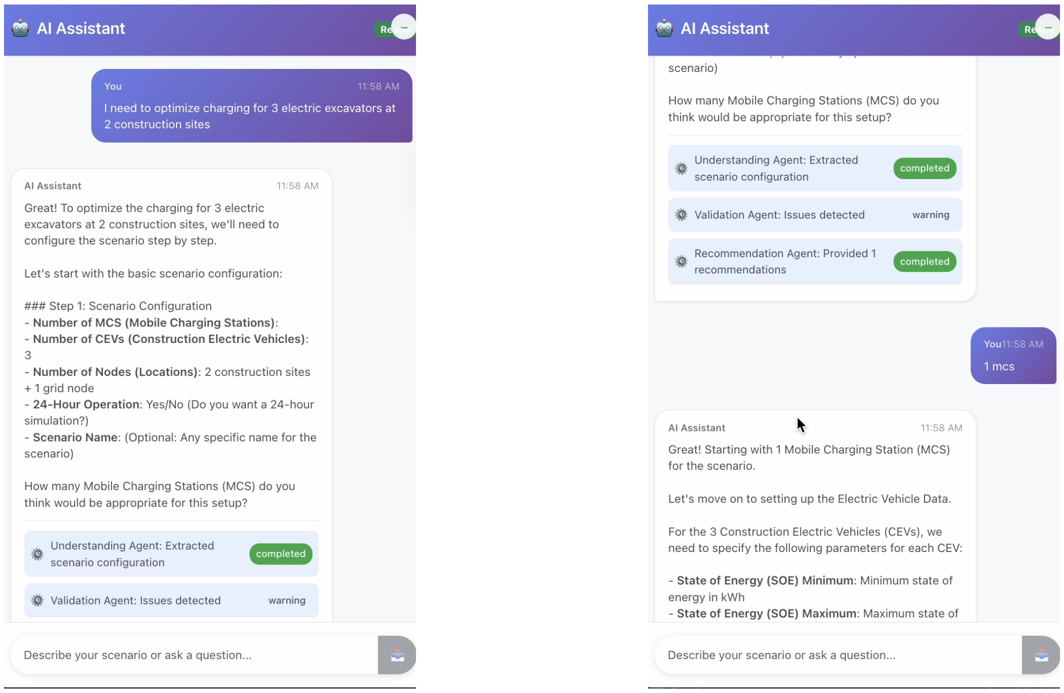
I'm your AI assistant for configuring mobile charging station optimization scenarios. I can help you:

- Configure scenarios with natural language
- Set up vehicle and location parameters
- Optimize charging schedules
- Analyze results and provide insights

Try saying: "I need to optimize charging for 3 electric excavators at 2 construction sites"

Describe your scenario or ask a question...

Figure 4: Landing page and AI Assistant panel in the demo UI.



You 11:58 AM
I need to optimize charging for 3 electric excavators at 2 construction sites

AI Assistant 11:58 AM
Great! To optimize the charging for 3 electric excavators at 2 construction sites, we'll need to configure the scenario step by step.

Let's start with the basic scenario configuration:

Step 1: Scenario Configuration
- **Number of MCS (Mobile Charging Stations)**:
- **Number of CEVs (Construction Electric Vehicles)**: 3
- **Number of Nodes (Locations)**: 2 construction sites + 1 grid node
- **24-Hour Operation**: Yes/No (Do you want a 24-hour simulation?)
- **Scenario Name**: (Optional: Any specific name for the scenario)

How many Mobile Charging Stations (MCS) do you think would be appropriate for this setup?

Understanding Agent: Extracted scenario configuration — completed

Validation Agent: Issues detected — warning

Describe your scenario or ask a question...

scenario)

How many Mobile Charging Stations (MCS) do you think would be appropriate for this setup?

Understanding Agent: Extracted scenario configuration — completed

Validation Agent: Issues detected — warning

Recommendation Agent: Provided 1 recommendations — completed

You 11:58 AM
1 mcs

AI Assistant 11:58 AM
Great! Starting with 1 Mobile Charging Station (MCS) for the scenario.

Let's move on to setting up the Electric Vehicle Data.

For the 3 Construction Electric Vehicles (CEVs), we need to specify the following parameters for each CEV:

- **State of Energy (SOE) Minimum**: Minimum state of energy in kWh
- **State of Energy (SOE) Maximum**: Maximum state of

Describe your scenario or ask a question...

Figure 5: Early conversation turns: intent capture and scenario configuration.