Journal of Zhejiang University-SCIENCE A in press
www.jzus.zju.edu.cn; www.springer.com/journal/11582
E-mail: jzus_a@zju.edu.cn

Research Article
https://doi.org/10.1631/jzus.A2500629

A multi-agent collaborative optimization framework for integrated energy system scheduling

Boan QU[1*], Haoyu JIANG[2*], Songjie WANG[2], Zheng LUO[2], Xueru LIN[2,3], Xingtao TIAN[4], Jian LI[5], Xiaojie LIN[2,6,7✉], Wei ZHONG[1,2]

[1]*Polytechnic Institute, Zhejiang University, Hangzhou 310027, China*
[2]*College of Energy Engineering, Zhejiang University, Hangzhou 310027, China*
[3]*School of Information and Electrical Engineering, Hangzhou City University, Hangzhou 310015, China*
[4]*Key Laboratory of Cleaner Intelligent Control on Coal & Electricity, Ministry of Education, Taiyuan University of Technology, Taiyuan 030024, China*
[5]*School of Mechanical Engineering, Beijing Institute of Technology, Beijing 100081, China*
[6]*Shanghai Institute for Advanced Study, Zhejiang University, Shanghai 200120, China*
[7]*Changzhou Industrial Technology Research Institute of Zhejiang University, Changzhou 213000, China*

Abstract: Integrated energy systems (IES) improve energy efficiency through coordinated optimization of electricity, heat, cooling, and gas, yet their scheduling optimization involves multienergy coupling, multidevice coordination, and complex constraints. Conventional approaches rely on experts to manually construct optimization models, suffering from high technical barriers, time-consuming development cycles, and limited transferability. In recent years, large language models (LLMs) have shown promise in natural language understanding and code generation; however, their direct application to IES scheduling optimization still faces challenges, including requirement ambiguity, attention drift in long contexts, and high debugging costs. To address these issues, this study proposes MASEO, a multiagent collaborative framework for IES scheduling optimization. MASEO offers three main contributions. First, the optimization workflow is decomposed into four sequential stages—information acquisition, mathematical modeling, code implementation, and execution validation—each handled by a dedicated LLM agent operating in series. Second, a structured checklist is introduced to standardize the information collection process, complemented by an error-classification-based traceable repair mechanism that routes failures to the responsible upstream agent for targeted correction. Third, case studies on a Beijing data center IES and a German community IES validate the framework: the annualized cost deviation in the Beijing scenario is approximately 1.4%, and the total cost in the German scenario is consistent with the expert benchmark. Compared with a single-agent baseline, modeling accuracy improves by approximately 35% and the code execution pass rate increases from 63% to 94%; ablation studies further validate the independent contribution of each core mechanism; multibackbone and sensitivity experiments further provide configuration guidance for deployment. The results suggest that MASEO can help reduce the dependence of IES scheduling optimization on specialized modeling expertise, offering a reference pathway for the reliable deployment of large language models in energy system scheduling optimization.

Key words: Integrated energy systems; Optimal scheduling; Large language models; Multiagent systems

1 Introduction

1.1 Background and motivation

Against the backdrop of the global energy transition and carbon neutrality goals, integrated energy systems (IES) have emerged as a key pathway for improving energy efficiency through synergistic optimization of electricity, heat, cooling, and gas, offering notable advantages in economics, reliability, and environmental performance (Huang et al., 2023; Song et al., 2022; Liu et al., 2014).

However, IES dispatch optimization is highly complex due to multienergy coupling, multidevice coordination, and diverse constraints (see Fig. 1). Significant scenario heterogeneity exists across regions, and the problem demands interdisciplinary

✉ Xiaojie LIN, xiaojie.lin@zju.edu.cn
* The two authors contributed equally to this work
Xiaojie LIN, https://orcid.org/0000-0002-0829-1143

Received Nov. 30, 2025; Revision accepted June 10, 2026; Crosschecked

expertise spanning energy engineering, operations research, and computer science. Existing methods rely on experts to manually formulate optimization models and solve them using frameworks such as Oemof and PyPSA, suffering from high technical barriers, time-consuming development cycles, and limited transferability (Fan et al., 2022; Breyer et al., 2025). Additionally, the dynamic operating environment of IESs can invalidate existing model assumptions, while static modeling approaches struggle to adapt rapidly.

Recent advances in large language model (LLM)-based agent technologies offer a promising alternative (Lin et al., 2025). LLMs possess natural language understanding, code generation (Roziere et al., 2023; Li et al., 2023), and external tool invocation capabilities (Zeng et al., 2024; Du et al., 2024), enabling natural language-driven model construction and solving and thus reducing dependence on specialized expertise (Jiang et al., 2025a, 2025b, 2026). However, challenges remain, including requirement uncertainty from natural language ambiguity, attention drift in long contexts, high debugging costs, and insufficient process transparency.

To address these issues, this paper proposes MASEO (Multi-Agent System for Energy Optimization). The framework decomposes dispatch optimization into four stages—information collection, mathematical modeling, code implementation, and execution verification—completed by specialized agents to reduce the single-process information load and improve task focus and traceability. A guided interaction mechanism ensures complete and accurate information acquisition, while a traceable local repair mechanism classifies errors and routes them to upstream agents for targeted correction.

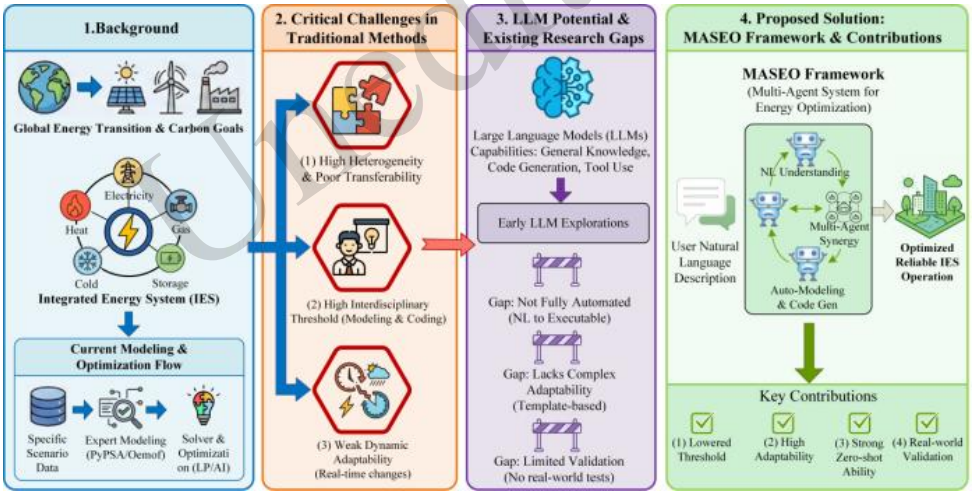

Fig. 1 The limitations of current scheduling methods for IESs

1.2 Literature review and research gap

In the domain of traditional optimization modeling and solving, extensive research has explored scheduling optimization across diverse IES configurations, spanning industrial parks, data centers, and community energy systems (Zhu et al., 2025; Yang et al., 2024). The emergence of open-source frameworks such as oemof and PyPSA has improved modeling efficiency and reduced manual programming effort by providing modular component support (Hilpert et al., 2018; Di Bella et al., 2024). However, given the significant heterogeneity among IESs in terms of device composition, energy topology, and operational constraints, optimization models are highly scenario-specific and difficult to transfer across applications. Each new scenario requires domain experts to reconduct system analysis, model construction, and code implementation from scratch, resulting in high technical barriers and lengthy development cycles that constrain the broader adoption of optimization technologies across diverse energy system deployments.

In the domain of LLM-based optimization modeling and solving, research has pursued two

directions. The first employs multiagent systems to simulate heuristic optimization processes as a substitute for conventional solvers (Liu et al., 2024; Wang et al., 2025; Yang et al., 2023), but such approaches have been validated only on simple combinatorial problems such as the traveling salesman problem and fall significantly short of mature solvers in solution accuracy, scalability, and real-time performance, rendering them unsuitable for engineering-grade IES dispatch. The second retains conventional solvers and leverages LLM natural language understanding and code generation capabilities to assist with modeling—including optimization formalization from natural language descriptions (Jin et al., 2023), power system simulation code generation (Jia et al., 2024), and LLM+ReAct-based agents for home energy management (Michelon et al., 2025)—reducing knowledge barriers and manual modeling workload to a certain extent. However, these efforts have focused on relatively simple, single-energy or small-scale systems and have not been extended to engineering-grade IES scheduling involving multienergy coupling and complex constraints (Fig. 2).

At a fundamental level, the widespread adoption of scheduling optimization technology faces a persistent core barrier: regardless of the tools employed, users must bear substantial learning and implementation costs for each new scenario—understanding problem structures, constructing mathematical models, and debugging optimization code. These demands on domain expertise and time investment prevent optimization technologies from reaching a broader range of users and heterogeneous energy system configurations (Jin et al., 2023; Michelon et al., 2025). While applying LLMs or single LLM-based agents to simple optimization tasks has demonstrated potential for lowering modeling barriers, the success rate remains low when facing IES scheduling—a multistage engineering problem involving multienergy coupling and complex constraints—and users still require considerable time for manual debugging and intervention. How to construct a reliable end-to-end pipeline from requirement acquisition and mathematical modeling to code execution in a human—AI collaborative manner remains an open and pressing challenge. To this end, this paper proposes MASEO, addressing this challenge through multiagent task decomposition, a structured checklist, and an error tracing and repair mechanism.

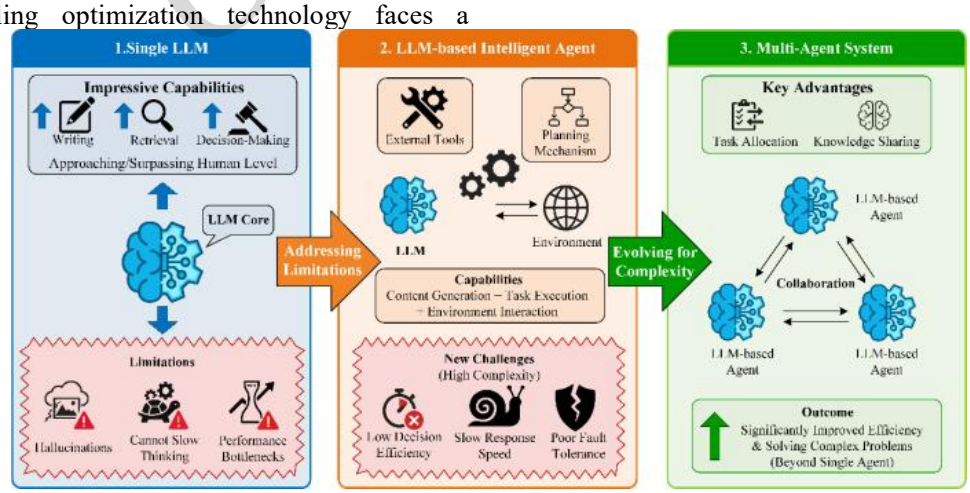

Fig. 2 Evolution of LLM-based approaches toward multi-agent systems

1.3 Contributions and paper organization

(1) A multiagent collaborative framework, MASEO, is proposed for IES dispatch optimization. Four specialized agents-problem formulation, model construction, code generation, and execution feedback-collaboratively transform natural language requirements into executable optimization workflows, addressing the complexity and expertise dependence of traditional approaches.

(2) A reliability assurance mechanism is proposed, incorporating a structured checklist for guided information collection and an error classification-based traceable repair mechanism for

iterative correction. Physical feasibility is verified through energy balance checks and equipment boundary validation.

(3) Case studies on a Beijing Data Center IES and a German Community IES validate the framework. MASEO completes scheduling optimization at the hourly level through human–machine interaction, improving modeling accuracy by approximately 35% and the code execution pass rate from 63% to 94% over a single-agent approach. Multibackbone and sensitivity experiments further confirm adaptability and provide configuration guidance for deployment.

2 Methodology

2.1 Basic definition of the IES optimization problem

A general optimization problem can be expressed in the following mathematical form:

$$\begin{aligned} &\min_{x\in X} f(x;\theta), \\ &\text{s.t. } g_i(x;\theta) \le 0, i = 1,2,\ldots,m, \\ &\quad h_j(x;\theta) = 0, j = 1,2,\ldots,n, \end{aligned} \tag{1}$$

where $x \in X \subseteq \mathbb{R}^n$ is the vector composed of all decision variables of the problem, X is the feasible region defined by physical constraints, and the parameter set θ includes objective function coefficients, constraint parameters, and system configuration information.

In IES dispatch optimization, $f(x;\theta)$ represents the performance index to be minimized (e.g., operational cost or carbon emissions), $g_i(x;\theta) \le 0$ describes operational boundary conditions such as equipment output limits and energy storage SOC bounds, and $h_j(x;\theta) = 0$ corresponds to physical conservation relationships such as energy bus power balance equations. The solution process comprises two stages: (1) a modeling stage that abstracts system information into a mathematical optimization model and (2) a solution stage that translates the model into executable code and invokes solvers to obtain optimal dispatch decisions. Both stages rely heavily on specialized knowledge, and the manual workflow is time-consuming and difficult to scale across diverse engineering scenarios, constraining the widespread adoption of IES optimization technologies.

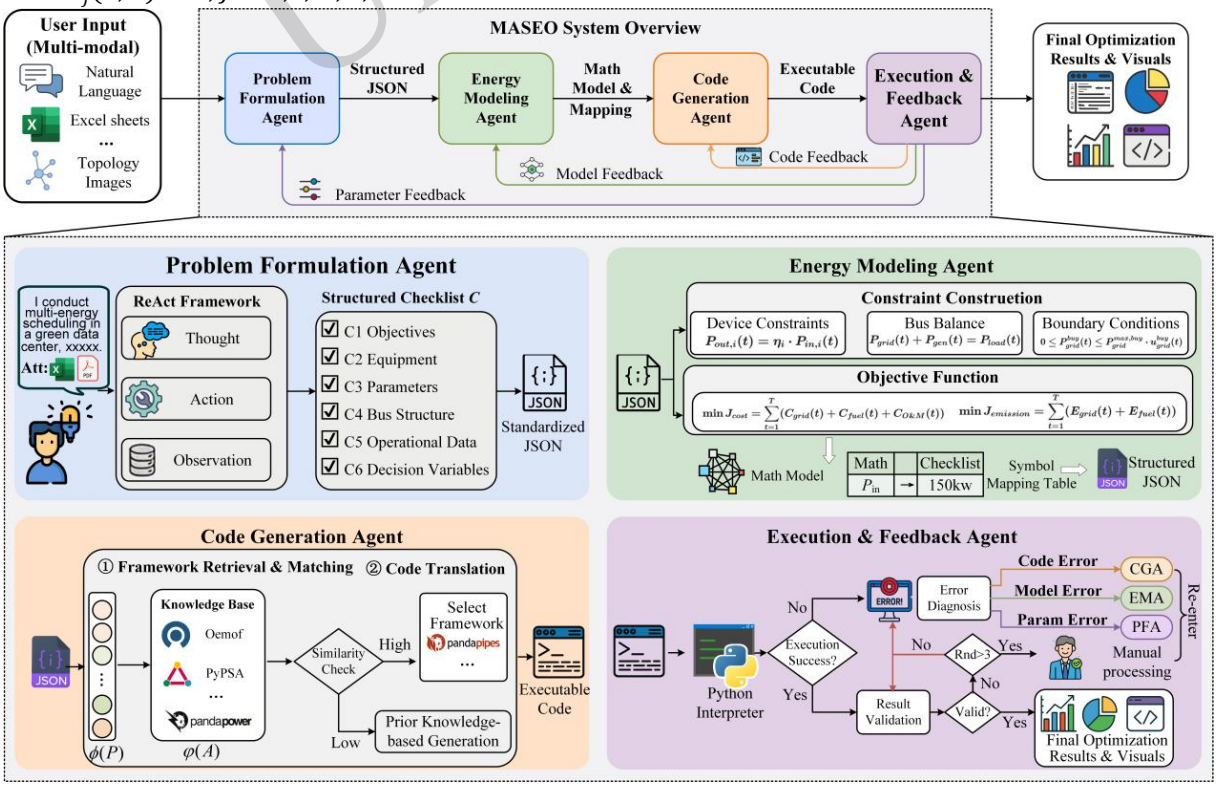

Fig. 3 MASEO system overview

2.2 The proposed MASEO framework

To automate this two-stage process, MASEO is constructed based on three design principles: task decomposition for reducing information overload and ensuring error traceability, guided interaction for mitigating requirement uncertainty and improving information completeness, and human-in-the-loop mechanisms for process transparency and reliability. As illustrated in Fig. 3, four sequential specialized agents operate within independent context spaces, exchanging information through structured process files, including a checklist, mathematical model, solution code, and error traceability table. Task type labels (generation or repair) distinguish initial generation from local revision. The agents collaboratively complete information collection, mathematical modeling, code generation, and execution verification, enabling end-to-end automation from natural language input to dispatch optimization results. Specifically, the PFA collects system information through multiround human–AI dialog and outputs a structured checklist; the EMA constructs the mathematical optimization model based on the checklist; the CGA transforms the model into executable Python optimization code; and the EFA executes the code, validates physical feasibility, and triggers targeted repair upon failure. Human–AI interaction is concentrated in the PFA stage; subsequent agents proceed autonomously in sequence. Operational workflows are illustrated in Fig. 4 The specific procedures are as follows:

(1) The user submits a dispatch requirement (Step 1a). The Problem Formulation Agent (PFA) interprets user intent and dynamically updates the checklist within the dialog history (Step 1b), then returns the current status and identifies missing elements (Step 2a), sending status confirmation and targeted inquiries to the user (Step 2b).

(2) The user confirms and supplements information accordingly (Step 3a). Steps 1b–3a iterate until the user confirms that the checklist is complete and accurate. The finalized checklist is transmitted to the Energy Modeling Agent (EMA) (Step 3b).

(3) The EMA retrieves from the historical model repository using equipment set and bus structure as indices (Steps 4a–4b), loading matched models as references or returning an empty set. Based on the checklist and retrieval results, the EMA constructs the mathematical optimization model and symbolic mapping table and then transmits them to the code generation agent (CGA) (Step 5a).

(4) The CGA retrieves from the optimization framework knowledge base (Steps 5b–6a), selects the most suitable framework, and generates executable Python optimization code, which is transmitted to the Execution & Feedback Agent (EFA) (Step 6b).

(5) The EFA executes the code and performs physical feasibility verification (Steps 7a–7b). If successful, dispatch visualizations are generated, and the model repository is updated (Steps 8a–8b). If verification fails or errors occur, an error traceability report is returned to the corresponding upstream agent for localized repair (Steps 9a–9b). If verification still fails after three iterations or an unrepairable error is encountered, the process terminates and manual intervention is requested (Step 9c).

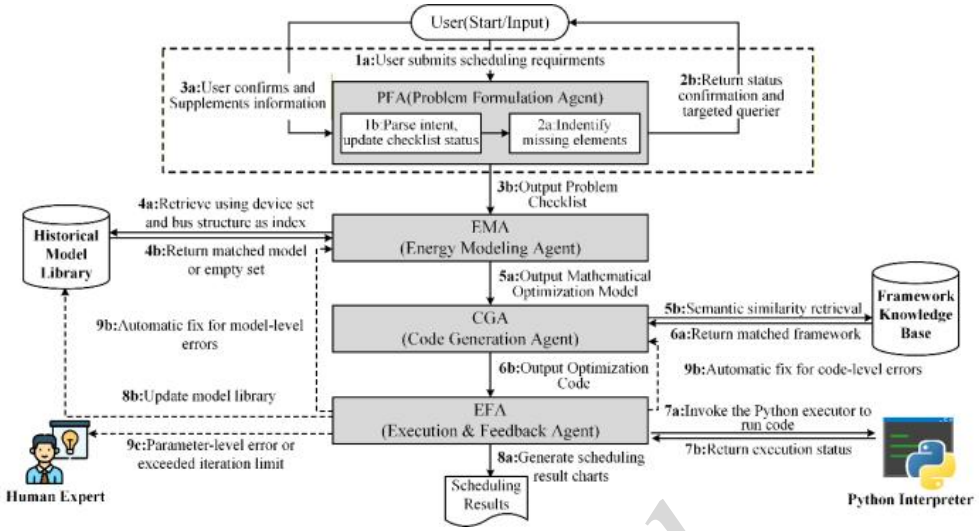

Fig. 4 Workflow diagram of the MASEO framework

2.3 Agent composition of the MASEO framework

Among the four agents, the PFA and EFA require iterative multiround reasoning and tool invocation and therefore adopt the ReAct framework; the EMA and CGA perform single-pass reasoning with tool-augmented chain-of-thought, where tools such as retrieval and file operations are invoked as part of the reasoning process but without iterative loops.

(1) Problem formulation agent

The PFA collects complete system information for IES dispatch optimization through multiround user interaction. Multiple file parsing tools are designed to support information extraction from various file formats (Appendix S1.1). A structured checklist C guides the information acquisition process, containing six core elements:

$$C = \{C_1, C_2, C_3, C_4, C_5, C_6\}. \quad (2)$$

Corresponding to optimization objective C_1, system equipment C_2, equipment parameters C_3, bus structure C_4, operational data C_5, and decision variables C_6. Among them, C_5 only stores the path references of time series data files and the statistical features extracted by the parsing tools. The state of C is implicitly maintained in the dialog history H_{PFA}, from which the PFA reconstructs the completion status of each element, identifies missing items, and plans subsequent actions.

As shown in Algorithm 1, the PFA adopts a nested loop structure. The outer loop is triggered by user input, where the PFA reasons over H_{PFA}, to identify missing elements and plan actions. If file parsing is needed, the inner loop automatically executes tool invocation and continuous reasoning until a user response is generated. Whenever any element C_i is updated, the PFA presents results and requests explicit user confirmation. Unconfirmed information is excluded from the valid state. Once all elements are confirmed, the PFA outputs the structured checklist via the file writing tool and transfers it to the EMA.

Algorithm 1 PFA ReAct Information Collection Loop

Input: System prompt S (including the checklist template and confirmation rules), tool set T

Output: The verified checklist.json

$H \leftarrow [\ ]$; confirmed $\leftarrow$ False

while confirmed = False **do** ▷ Outer loop: starting from the user input.

  $u \leftarrow$ UserInput(); $H \leftarrow H \oplus u$

  repeat ▷ Inner loop: tool invocation chain.

    (thought, action, response) $\leftarrow$ LLM(S, H)

    if action.type = tool_call **then**

      obs $\leftarrow T$ [action.name](action.args)

      $H \leftarrow H \oplus$ (thought, action, obs)

  until *action.type* $\neq$ *tool_call* ▷ If no tool is invoked, the system outputs a response.

$H \leftarrow H \oplus$ response

if action.type = file_writer **then** confirmed ← True

end while

(2) Energy modeling agent

The EMA takes a structured checklist as input and transforms it into a computationally solvable mathematical optimization model. Unlike the PFA, the EMA addresses inference-intensive tasks with clearly defined and complete input. The tool configuration is shown in Appendix S1.2.

To reduce redundant reasoning overhead, the EMA first retrieves historical models from the repository using the equipment set and bus structure as indices. If matched, the most structurally similar historical model M^* is loaded into the reasoning context $H_{\mathrm{EMA}} = \{C, M^*\}$ for joint model construction; otherwise, $H_{\mathrm{EMA}} = \{C\}$, and reasoning proceeds based solely on the checklist.

The EMA inference comprises two stages. In the mathematical expression generation stage, operational and energy conversion constraints are generated based on device types in C_2; power balance equations are constructed according to the bus topology in C_4; and the objective function is formulated per the optimization objectives in C_1. In the symbol mapping stage, the correspondence between checklist parameters and model symbols is established: decision variables in C_6 map to optimization variables, device parameters in C_3 to known scalars, and time-series data in C_5 to time-varying parameters referenced via file paths and column names. The resulting optimization model is output and passed to the CGA.

(2) Code generation agent

The CGA transforms the mathematical optimization model from the EMA into executable Python solving code, essentially converting decision variables, objective functions, and constraints into code expressions while selecting an appropriate solution method. The tool configuration is shown in Appendix S1.3.

To mitigate issues such as improper framework selection and incorrect interface calls, a structured knowledge base is constructed from Python-oriented optimization frameworks (e.g., Pyomo, Oemof, PyPSA, Calliope) to support retrieval-augmented generation (RAG). Solver invocation rules (e.g., GLPK, Gurobi) are incorporated into the system prompt to constrain code generation.

Specifically, the CGA represents the coding semantics of the current problem as a vector $\boldsymbol{\phi}(P)$, and computes the cosine similarity with the semantic vectors $\boldsymbol{\phi}(A)$ of candidate frameworks in the knowledge base:

$$S(P, A) = \frac{\boldsymbol{\phi}(P) \cdot \boldsymbol{\phi}(A)}{|\boldsymbol{\phi}(P)||\boldsymbol{\phi}(A)|}. \quad (3)$$

CGA selects the framework with the highest score,

$$A^* = \arg\max_A S(P, A), \quad (4)$$

as the reference framework for code generation. When the highest score is lower than the preset threshold τ, the current problem is considered to have insufficient correspondence with existing framework knowledge. In this case, the code generation process does not incorporate retrieval results and instead constructs the solving code solely based on the mathematical optimization model.

Finally, CGA invokes the file-writing tool to output the generated code and passes it to EFA for execution and validation.

(3) Execution & feedback agent

The EFA executes optimization code and performs state assessment, error diagnosis, feedback-based repair, or result output (Madaan et al., 2023; Diallo et al., 2025). The tool configuration and reasoning workflow are provided in Appendix S1.4. Five error categories and corresponding routing strategies are defined to ensure traceability (Table 1). EFA executes optimization code through the code execution tool within the user's preconfigured local Python environment; the LLM does not have permission to install dependencies autonomously.

Table 1 EFA Error classification and routing strategies

Error Type	Trigger Condition	Routing Target	Repair Mode
Code-level Error	Code syntax exceptions or	CGA	Perform targeted revisions to the

	incorrect API calls to the modeling framework		optimization solving code and append a summary of the changes
Model-level Error	Structural defects in the model cause the solver to return infeasible or unbounded results	EMA → CGA	EMA locally corrects the mathematical optimization model, and CGA synchronizes the code update according to the change summary
Parameter-level Error	Parameter out-of-range values, dimension mismatches, or unit inconsistencies	EMA → CGA	Same as the model-level error repair process
Environment-level Error	Solver not installed or required runtime dependencies missing	—	Archive process files, terminate the workflow, and return environment configuration recommendations
Information-deficiency Error	Critical elements missing or logical inconsistencies in the problem checklist	—	Archive process files, terminate the workflow, and return a defect localization report

Upon successful execution, the EFA validates physical feasibility by examining device output boundaries and energy conservation constraints. If passed, visualization charts are generated, and process artifacts are archived, with the model metadata written to the historical repository for retrieval and reuse. If execution fails or validation is not satisfied, the error type, message, and relevant file references are recorded in an error tracing table. The task is routed to the corresponding upstream agent for targeted local revision, with modification summaries recorded to enable downstream synchronization, avoiding full regeneration overhead.

The maximum number of automatic repair iterations is three. If validation still fails or environment-level/information-deficiency errors occur, the EFA archives all process artifacts, notifies the user, and terminates the workflow for manual intervention

3 Case Study

To evaluate the practical effectiveness of the MASEO framework in the scheduling optimization of IESs, this study selects the DCIES in Beijing and the CIES in Germany as validation cases.

3.1 Case #1: Data center integrated energy system

The Beijing DCIES features hydrogen energy utilization and cascade recovery of data center waste heat, forming a low-carbon multienergy supply system (Fig. 5). The data center has a rated IT capacity of 20 MW (~95% utilization, ~19 MW actual load), providing electricity, district heating, and industrial steam services. The supply side integrates photovoltaic, wind power, and hydrogen fuel cells. Waste heat is recovered through cascade utilization: low-grade waste heat (20 - 40°C) is upgraded by a low-temperature heat pump and further converted to steam by a high-temperature steam heat pump. Cooling is jointly provided by electric chillers, absorption chillers, cooling towers, and cold energy recovered from liquid hydrogen regeneration. Thermal and cold storage units smooth load fluctuations. The equipment parameters are listed in Table 2.

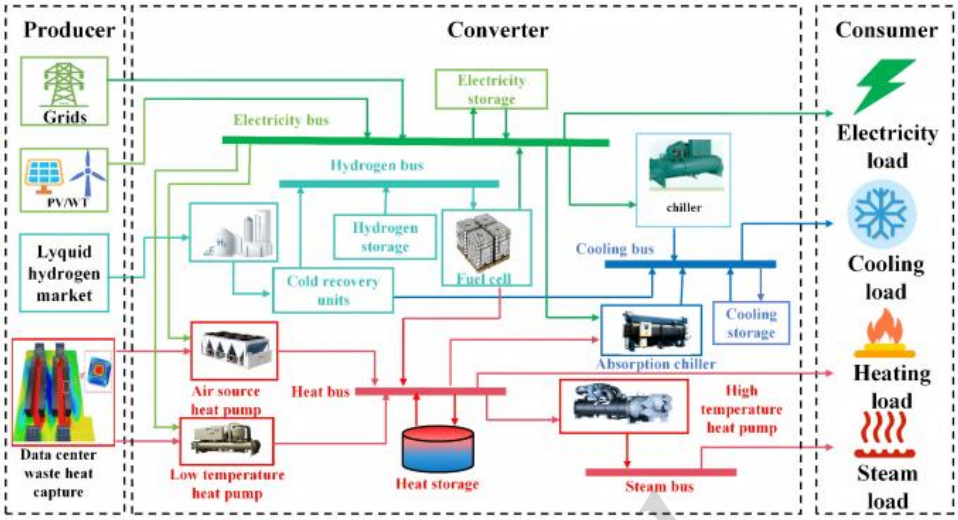

Fig. 5 Energy flow diagram of the system (Wang et al., 2025)

Table 2 Equipment capacity and efficiency parameters

Equipment	Capacity (MW)	Energy efficiency/COP
Photovoltaic	15.56	20%
Wind Turbine	20.00	20%
Hydrogen Fuel Cell	20.15	ele/heat: 60%/35%
High-temperature Steam Heat Pump	7.746	3.5
Low-temperature Heat Pump	23.91	4.2
Electric Chiller	35.00	2.87
Absorption Chiller	3.41	0.75
Cooling Tower	40.00	98%
Thermal Storage	19.96	98%
Cold Storage	2.62	98%

The operational dataset comprises hourly data for 2022 (8,760 steps), covering electricity, cooling, heating, and steam demands, along with meteorological variables. A typical-day clustering method extracted 10 representative days, with clustering parameters and time-of-use electricity tariffs detailed in Appendix S2.1 (Fig. 6). The carbon emission cost is approximately \$0.008/kWh (grid factor: 0.5703 $kgCO_2$/kWh, carbon price: \$13.47/ton). The liquid hydrogen price is \$0.075/kWh; the external heating, steam, and cooling prices are \$0.022, \$0.056, and \$0.069/kWh, respectively.

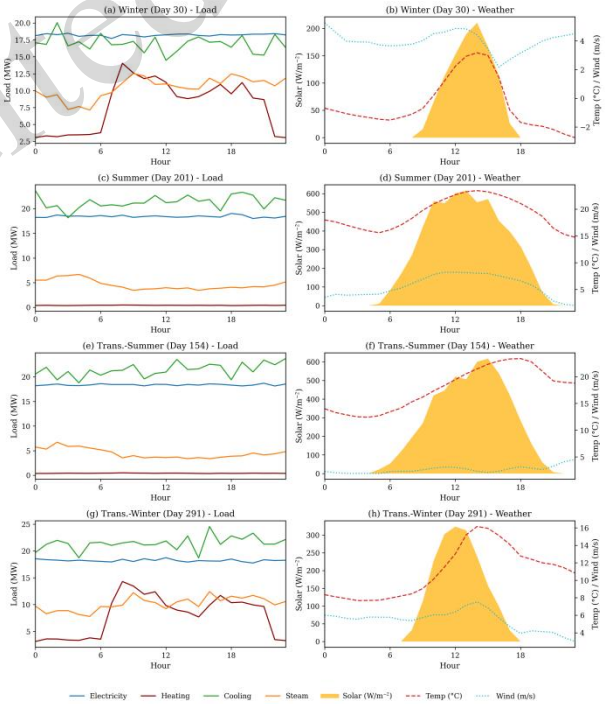

Fig. 6 Hourly load profiles and meteorological conditions for four typical days: (a)(b) Winter, (c)(d) Summer, (e)(f) Transitional summer, and (g)(h) Transitional winter

3.2 Case #2: Integrated community energy system

The German CIES is a CCHP system centered on a gas turbine, serving the electricity, heating, and cooling demands of an industrial community (Fig. 7). The supply side integrates photovoltaic and wind power, with a gas turbine as the core cogeneration unit. Waste heat is recovered by an absorption chiller for cooling. An electric heat pump and electric chiller

supplement heating and cooling. Battery, thermal, and cold storage enable peak shaving and load shifting, with grid connections ensuring energy balance. The equipment parameters are listed in Table 3.

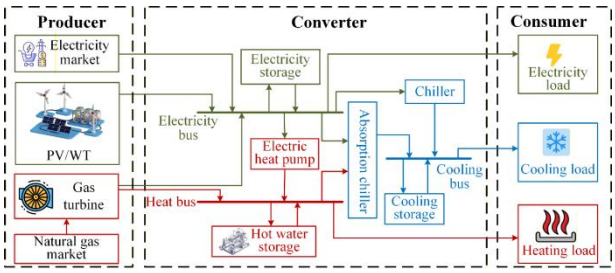

Fig. 7 Structural diagram of the CIES

Table 3 Equipment capacity and efficiency parameters

Equipment	Capacity (MW)	Energy efficiency/COP
Photovoltaic	1.234	20%
Wind Turbine	1.366	20%
Gas Turbine	0.715	ele/heat: 33%/50%
Electric Heat Pump	0.803	4.44
Electric Chiller	0.277	2.87
Absorption Chiller	0.426	0.75
Battery Storage	0.859	85.5%
Thermal Storage	1.610	81%
Cold Storage	0.947	81%

Operational data consist of hourly time series for the full year, with meteorological data from NASA MERRA-2 (Gelaro et al., 2017) and load data from the Scientific Data dataset (Priesmann et al., 2021). Ten typical days were extracted by clustering, with clustering parameters and electricity pricing details provided in Appendix S2.2 (Fig. 8). Electricity pricing follows a two-part tariff: capacity charge €114.29/(kW·a) and energy charge €0.0025/kWh. The natural gas price is €0.0286/kWh.

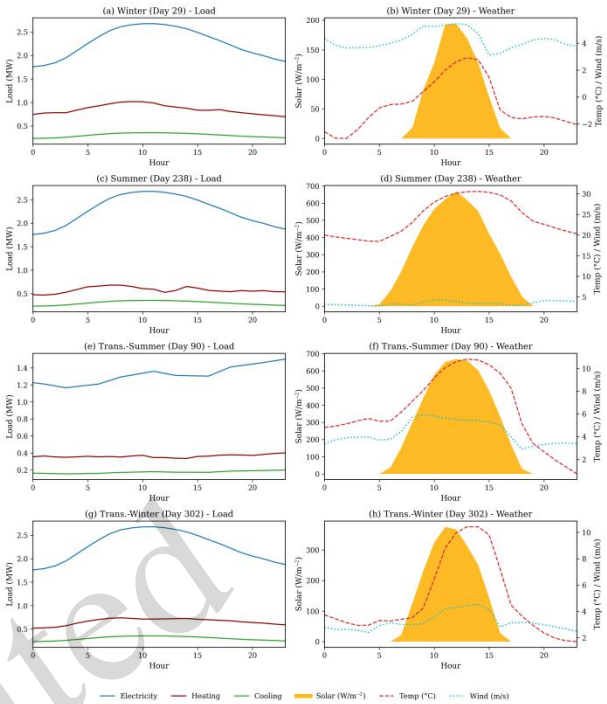

Fig. 8. Hourly load profiles and meteorological conditions for four typical days: (a)(b) Winter, (c)(d) Summer, (e)(f) Transitional summer, and (g)(h) Transitional winter

3.3 Evaluation metrics

The framework is assessed across four dimensions. Expert-annotated references serve as benchmarks, with an LLM-as-Judge approach for scoring to avoid misclassification by rule-based methods. The evaluation results are reviewed and confirmed by domain experts.

(1) Optimization problem modeling accuracy

Measuring the capability to extract information and construct mathematical formulations:

$$S_{\text{modeling}} = \frac{1}{N}\sum_{i=1}^{N} s_i,\ s_i \in [0,1]. \tag{5}$$

Information collection completeness includes equipment completeness s_1, topology accuracy s_2, completeness of parameters and variables s_3, and objective function accuracy s_4. Model completeness includes constraint completeness s_5, accuracy of model representation s_6, and accuracy of parameter mapping s_7. Scores are interpreted in three intervals: $[0.60,0.70)$ indicates system-level structural errors such as missing bus connections or mismatched equipment; $[0.70,0.85)$ indicates correct topology but missing local constraints or parameter mapping errors;

[0.85,1.00) indicates high accuracy with only minor differences.

(2) Code execution pass rate

The code execution pass rate is used to reflect the capability of the framework to generate accurate and executable code. The following metrics are defined:

First-pass execution rate (FER):

$$R_1=\frac{N_{success}^{(1)}}{N_{total}}. \quad (6)$$

Postrepair execution rate (PER):

$$R_r=\frac{N_{success}^{(\leq 3)}}{N_{total}}. \quad (7)$$

(3) Solution accuracy

Both cases are LP problems with guaranteed global optima. Using the expert model solution y^* as a benchmark:

$$\delta=\left|\frac{|y-y^*|}{|y^*|}\right| \quad (8)$$

where y is the framework's objective value; a smaller δ indicates higher consistency.

(4) **Framework economic efficiency:** evaluated by token consumption and cost per task.

3.4 Case validation

To evaluate usability for low-barrier users, a first-year energy engineering graduate student without IES scheduling optimization experience independently completed both case tasks using only system documentation and the interactive interface, without access to reference model implementation details. The equipment parameters were consistent with those in Sections 3.1 and 3.2. Claude-Sonnet-4.5 was used as the base model (Table 4).

Table 4 LLM Configuration for the Case Validation Experiments

Configuration	Value
Model	Claude-sonnet-4.5
Context window	200K tokens
Max output tokens	16384
Temperature	0.3
Top-p	0.95

3.5 Framework performance evaluation

An experimental dataset of 13 scheduling optimization task instances was constructed from multiple equipment capacity configurations derived from the planning results of both cases (details in Appendix S2). LLMs simulated realistic user input scenarios with partial information provision capability. Unless otherwise specified, all experiments adopted a unified model configuration and default parameters.

(1) Comparative Experiments: To evaluate the overall performance and backbone adaptability, experiments were conducted on several mainstream models under unified hyperparameters (Table 5). A single-agent ReAct baseline using Claude-Sonnet-4.5 with a single context window completing the entire pipeline was constructed for comparison. Each configuration was run five times with averaged results.

Table 5 Pricing Information for Backbone Models

Model	Input Price ($/M tokens)	Output Price ($/M tokens)
Llama-4-Scout-17B	0.60	2.40
Qwen3-32B	1.20	5.148
DeepSeek-R1	1.40	5.60
DeepSeek-V3	0.80	3.20
Qwen3-235B-A22B	1.95	8.365
Qwen2.5-72B-Instruct	2.50	7.50
GPT-4.1	2.00	8.00
Claude-Sonnet-4.5	3.00	15.00

4 Experimental results

4.1 Framework performance evaluation

API pricing followed the Claude-Sonnet-4.5 rate (Table 5). As this study does not involve planning, the annual investment cost was set identical to the expert model.

(1) Case #1: Data center integrated energy System

Table 6 compares key economic indicators between MASEO and the expert model. The annualized total cost deviation is approximately 1.4% ($13.8309 million vs. $14.03 million). The grid electricity procurement cost decreased by 13.2%, the hydrogen purchase cost increased by 17.1%, and the waste heat recovery revenue increased by 10.4%. These differences stem from reasonable modeling

variations in cooling tower treatment: MASEO models it as an adjustable conversion device, granting greater scheduling flexibility. Consequently, the framework tends to reduce grid purchases and increase hydrogen consumption to lower carbon tax expenditures. Such differences fall within a reasonable strategy selection space, and experts can review or adjust cooling tower strategies according to actual scenarios.

The end-to-end task completion time was 2.5 hours, during which two errors occurred, both automatically corrected. The first, a CODE_ERROR caused by a mismatch between Excel column names and code variable names, was routed by the EFA to the CGA. Second, a MODEL_ERROR from a missing constraint rendering the cooling tower unavailable above 13°C was fed back to the EMA for constraint supplementation. The single-task cost was $1.334. Interaction statistics and token consumption are presented in Table 7.

Table 6 Performance in the Data Center Integrated Energy System

Cost		Expert model	MASEO
Annualized cost/(10^4 $)	Capital cost	718	718
	Grid	338	293.35
	Hydrogen	460	538.53
	Carbon tax	85	83.63
Surrounding heating cost/(10^4 $)	Heating	72	72.81
	steam	167	121.36
Data center waste heat revenue/(10^4 $)		-402	-443.91
Total cost/(10^4 $)		1403	1383.09

Table 7 Interaction statistics of the MASEO framework

Metric	Value
Total dialog turns	19
PFA dialog turns	11
EMA iterations	2
CGA iterations	3
EFA executions	3
Errors encountered	CODE_ERROR (1), MODEL_ERROR (1)
Token consumption and cost	input/output tokens/cost: 0.255 M/0.0378 M/$1.334

Fig. 9 presents the energy dispatch results for four typical days, with all energy carriers satisfying the bus balance constraints at every time step. The electricity load remains stable at 18–19 MW, with the hydrogen fuel cell providing a baseload (average 4.79 MW) and renewable output fluctuating seasonally (wind peak 15.08 MW in summer, PV peak 10.47 MW in transitional seasons). Cooling (average 20.30 MW) exhibits seasonal switching: the cooling tower dominates in winter (15.71 MW), while the electric chiller takes over in summer (17.40 MW). Heating peaks at 14.04 MW in winter and drops to 0.42 MW in summer, primarily from the low-temperature heat pump. Steam peaks at 12.53 MW in winter, mainly from the high-temperature steam heat pump.

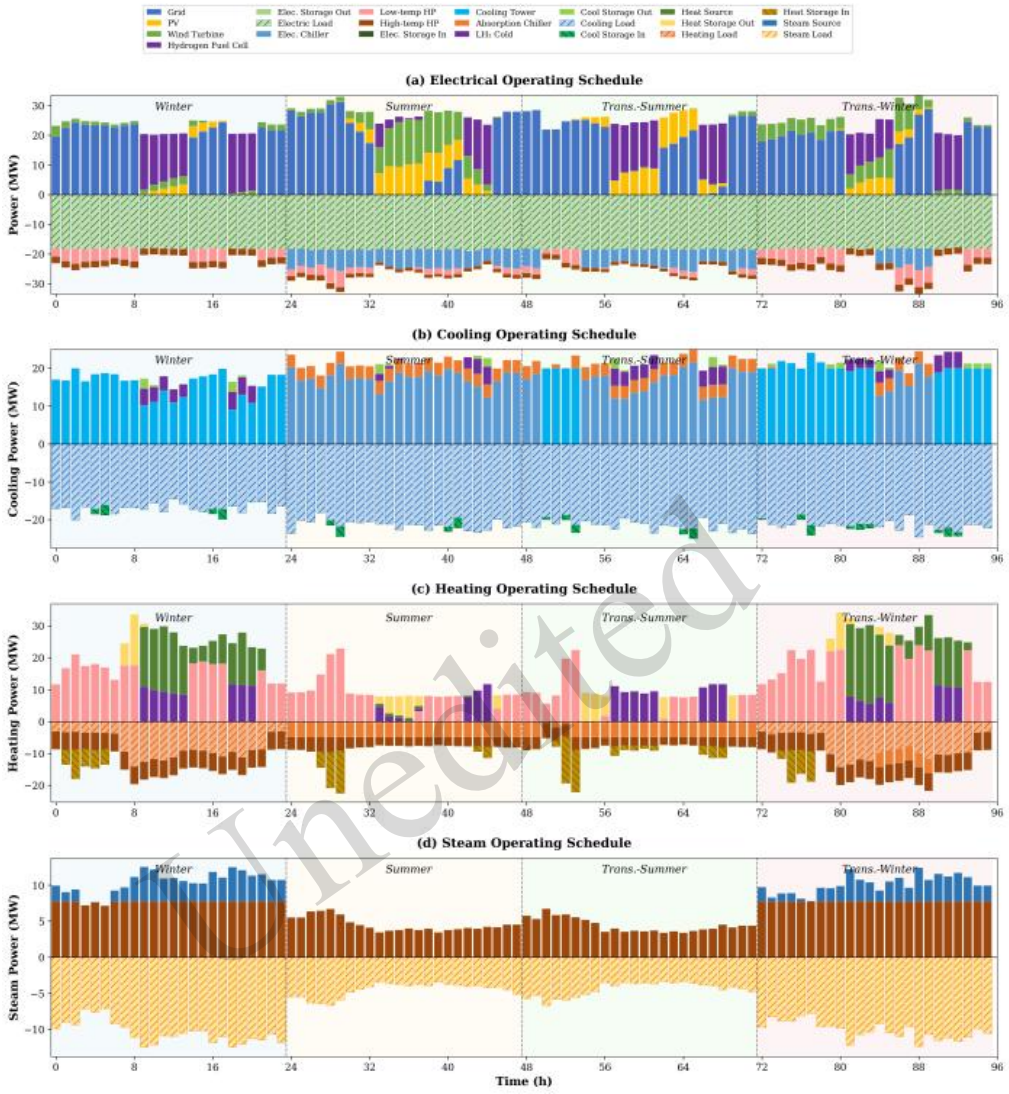

Fig. 9 Hourly energy balance schedules for four representative typical days. (a) Electrical balance, (b) cooling balance, (c) heating balance, and (d) steam balance. Positive values indicate energy supply, negative values (hatched) indicate demand

(2) Case #2: Integrated Community Energy System in Germany

Table 8 presents MASEO's results for the CIES. Due to the relatively simple system structure, MASEO generated executable code in a single attempt and passed physical feasibility checks. The annualized total cost is \$985,800, consistent with the expert model. Capacity costs account for the largest share (38.3%), aligning with the two-part tariff commonly applied in German industrial pricing. The single-task API cost was \$0.997, lower than Case #1. The end-to-end completion time was approximately 1.5 hours. The interaction statistics and token consumption are shown in Table 9.

Precision stamping tests were carried out several times under the traditional processing methods on steel materials (No. 35#). The test data is shown in Table 1(Calculated at an exchange rate of 1 EUR = \$1.16).

Table 8 Performance in the Community Integrated Energy System

Cost		MASEO
Annualized cost/(10^4 \$)	Capital cost	56.18
	Grid	4.17
	Gas	0.51
	Capacity	37.72
Total cost/(10^4 \$)		98.58

Table 9 Interaction statistics and token consumption data

Metric	Value

Total dialog turns	12
PFA dialog turns	9
EMA iterations	1
CGA iterations	1
EFA executions	1
Errors encountered	None
Token consumption and cost	input/output tokens/cost: 0.201 M/0.0263 M/$0.977

Fig. 10 illustrates the energy dispatch results for four typical days, with all carriers satisfying the bus balance constraints. The electricity load averages 2,033 kW (peak 2,678 kW) with a weekday–weekend pattern, primarily grid-supplied. Wind power contributes most in transitional winter (peak 371.4 kW, renewable penetration 14.2%); PV averages 115.5 kW in summer and 26.7 kW in winter. The gas turbine operates only on extreme cold days (-0.6°C, average 61.1 kW). Heating shows strong seasonality (winter 852.3 kW, summer 577.7 kW), with the electric heat pump as the core unit. Cooling (average 269.8 kW) features EC-AC complementary operation: EC dominates in winter (91%), AC alone in transitional winter, parallel in summer (AC 56%, EC 44%), and EC-led in transitional summer (65%).

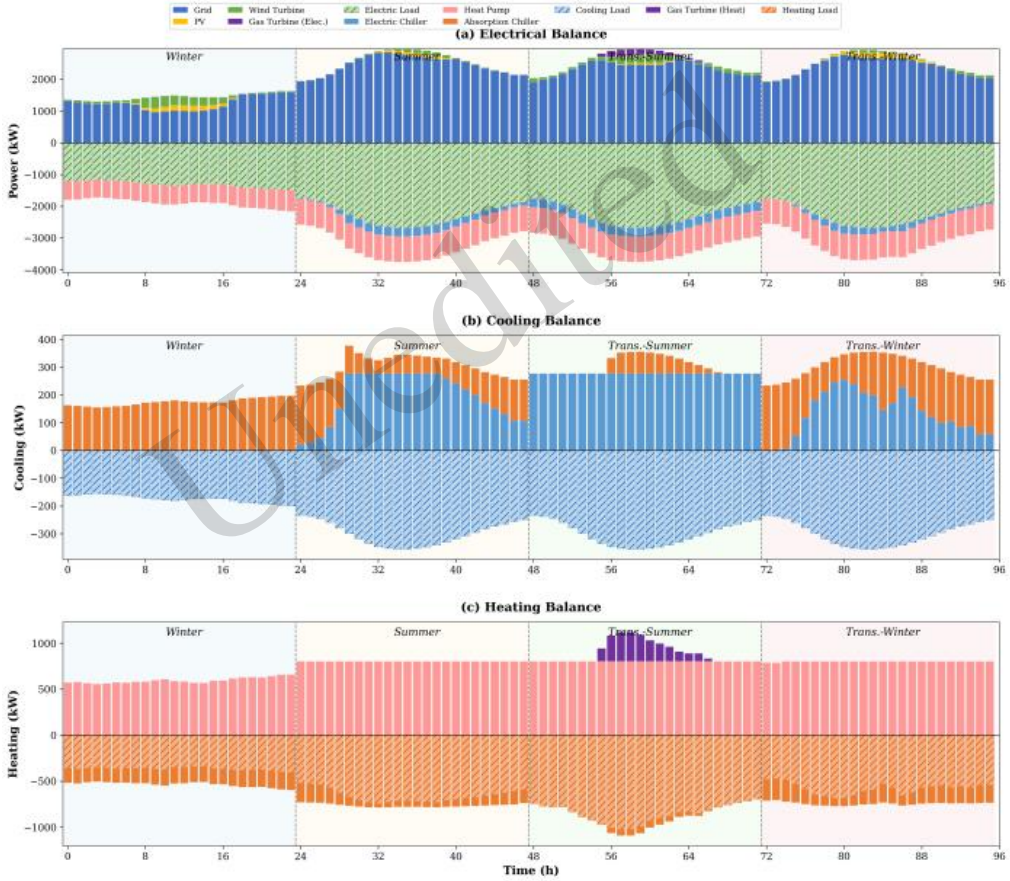

Fig. 10 Hourly energy balance schedules for four representative typical days. (a) Electrical balance, (b) cooling balance, (c) heating balance, and (d) steam balance. Positive values indicate energy supply, negative values (hatched) indicate demand

Case studies demonstrate that the MASEO framework compresses the scheduling optimization work originally performed by domain experts from days or weeks to within hours while maintaining result accuracy consistent with expert benchmarks. The automatic correction of two errors in the Beijing data center case further validates the effectiveness of the traceable repair mechanism, partially alleviating the low success rate and heavy manual debugging burden that characterize existing LLM-assisted approaches.

4.2 Comparative experiment results

(1) Baseline and ablation analysis

As shown in Fig. 11, the single-agent LLM+ReAct baseline yields failure rates of 40%/37% and S_{modeling} of 0.652/0.668 across the two scenarios.

The low accuracy reflects the difficulty of sustaining complete constraint representations over extended dialogs—critical parameters are progressively overwritten, and the resulting omissions propagate into modeling errors. The high failure rates indicate continued reliance on manual debugging, while the large variance reveals that unstructured requirement collection introduces substantial run-to-run inconsistency. MASEO improves S_{modeling} to 0.881/0.897 (variance: 0.021/0.018), reduces failure rates to 7%/6%, and narrows solution deviation from 5.84%/5.21% to 1.23%/0.95%, suggesting that the multiagent framework offers meaningful improvements for complex IES scheduling tasks.

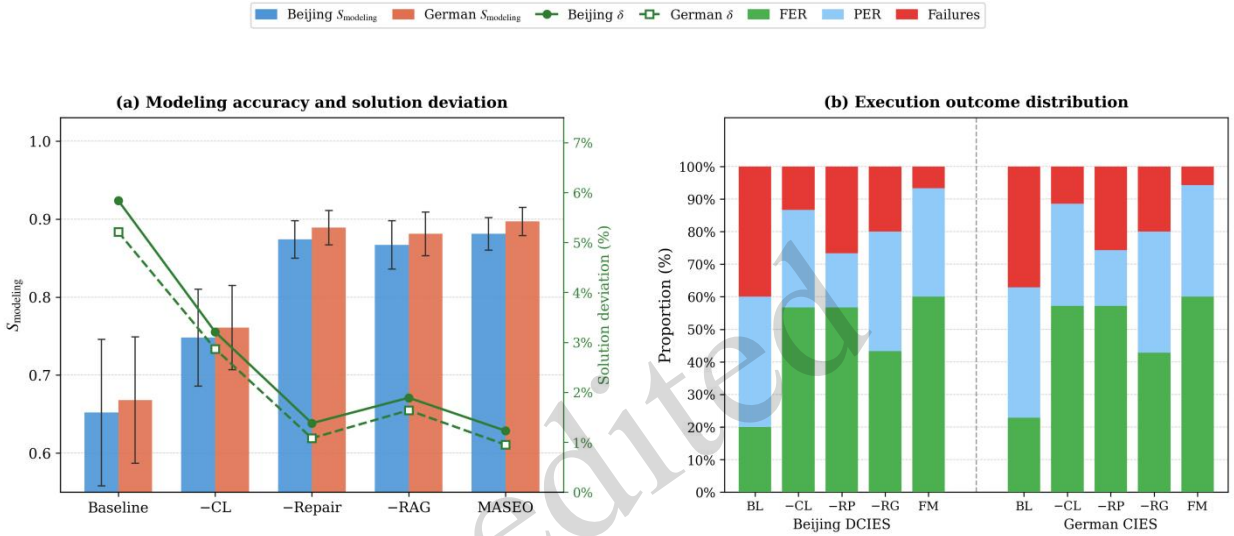

Fig. 11 Baseline and ablation comparison of MASEO across two scenarios. (a) Modeling accuracy and solution deviation; (b) execution outcome distribution. BL: Single LLM+ReAct baseline; −CL: w/o structured checklist; −RP: w/o traceable repair; −RG: w/o RAG retrieval; FM: full MASEO

The ablation results reveal the distinct role of each component. Removing the structured checklist (−CL) causes the largest drop in S_{modeling}(0.748/0.761) and widens solution deviation to 3.21%/2.87% with increased variance, while execution pass rates remain comparatively stable—confirming that the checklist primarily governs modeling quality and stability. Removing the repair mechanism (−RP) leaves S_{modeling} largely unchanged (0.874/0.889) but raises failure rates to 27%/26% and drops PER from 33%/34% to 17%; without error routing, the EFA regenerates blindly, and the same failure patterns recur. Removing RAG (−RG) mainly reduces FER from 60% to 43% with minimal effect on S_{modeling}. The three modules thus address modeling stability, execution reliability, and code quality as independent dimensions.

Table 10 reports token consumption and per-task cost. Full MASEO costs approximately $40.59 (Beijing) and $28.62 (German), with the difference driven by system complexity and repair iterations. Despite using a single context, the baseline accumulates comparable input tokens due to a long dialog history. Ablation variants incur slightly lower costs from reduced module inputs. Overall, the multiagent overhead remains modest and economically feasible for deployment.

Table 10 Token consumption and per-task cost across configurations

Configuration	Beijing DCIES			German CIES		
	Input (M tok)	Output (M tok)	Cost ($)	Input (M tok)	Output (M tok)	Cost ($)
Baseline	9.82	2.14	42.15	7.24	1.86	32.48
−CL	7.21	1.08	37.14	4.11	0.88	24.03
−Repair	7.52	1.02	37.85	4.44	0.85	25.37
−RAG	7.84	1.09	38.54	4.28	0.91	25.88

MASEO	8.03	1.10	40.59	4.96	0.92	28.62

(2) Backbone model comparison

As shown in Fig. 12, MASEO maintains a code execution pass rate (FER+PER) above 80% and S_{modeling}above 0.76 across all tested backbone models, demonstrating the framework's adaptability to different model configurations and providing a reference for model selection in practical deployment. Token consumption and per-task costs are reported in Table 11. Among closed-source models, Claude Sonnet-4.5 (M1) achieves the best overall performance (S_{modeling} 0.881 in Beijing, failure rate 7%), although at a relatively high per-task cost ($40.59/$28.62); GPT-4.1 (M2) delivers comparable results at $26.44/$18.39, offering a more favorable cost-performance ratio. Open-source large-parameter models (M3–M6) maintain stable performance with S_{modeling} exceeding 0.82 and failure rates of 10%–11%, remaining within an acceptable range relative to closed-source models; among these, DeepSeek-V3 (M4) incurs the lowest cost ($21.79/$13.56) while sustaining S_{modeling} of 0.826/0.836, demonstrating strong economic feasibility. The lightweight Qwen3-32B (M7) achieves an accuracy comparable to larger open-source models at $23.36/$16.60, making it suitable for resource-constrained deployments. Llama-4-Scout-17B (M8) performs relatively poorly, with S_{modeling} dropping to 0.758/0.764 and failure rates rising to 23%, suggesting that the framework imposes a minimum threshold on backbone model reasoning capability. Overall, Claude Sonnet-4.5 or GPT-4.1 is recommended when modeling accuracy is the primary concern; Qwen3-32B or DeepSeek-V3 offers a viable balance of performance and cost under resource constraints; and large open-source models are appropriate where both performance and deployment autonomy are needed.

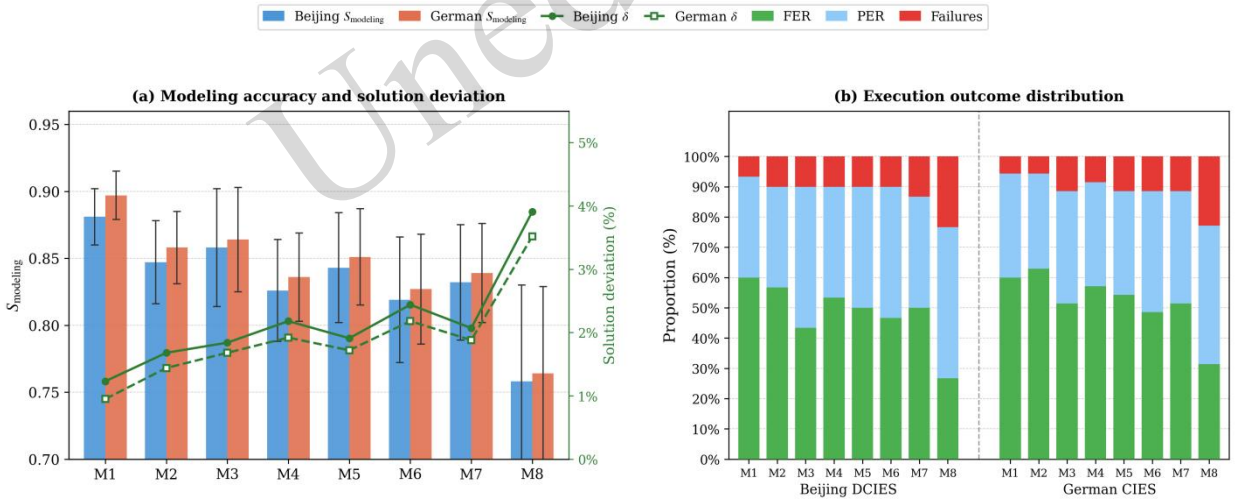

Fig. 12 Backbone model comparison under MASEO. (a) Modeling accuracy and solution deviation; (b) execution outcome distribution. M1–M8 denote Claude Sonnet-4.5, GPT-4.1, DeepSeek-R1, DeepSeek-V3, Qwen3-235B-A22B, Qwen2.5-72B-Instruct, Qwen3-32B, and Llama-4-Scout-17B, respectively

Table 11 Token consumption and per-task cost across backbone model configurations

Configuration	Beijing DCIES			German CIES		
	Input	Output	Cost ($)	Input	Output	Cost ($)
Claude Sonnet-4.5	8.03	1.10	40.59	4.96	0.92	28.62
GPT-4.1	5.42	0.98	26.44	3.82	0.71	18.39
DeepSeek-R1	6.11	1.41	27.57	5.12	1.24	22.41
DeepSeek-V3	5.11	1.01	21.79	3.48	0.78	13.56
Qwen3-235B	8.18	1.34	34.78	3.14	0.96	21.32
Qwen2.5-72B	6.07	1.24	32.16	4.28	1.02	25.46
Qwen3-32B	4.99	1.05	23.36	3.28	0.84	16.60

Llama-4-Scout	5.14	1.15	26.00	3.42	0.88	13.19

4.3 Sensitivity analysis experimental results

As shown in Fig. 13 and Table 12, the code execution pass rate (FER+PER) exceeds 80% across all temperature settings in both scenarios, indicating that MASEO's execution reliability is robust to temperature variation and reflecting the stabilizing role of the traceable repair mechanism. In contrast, temperature has a more pronounced effect on modeling quality, with Smodelingranging from 0.71 to 0.91. Within the lower portion of this range, the model generally preserves the correct system topology but may exhibit local constraint omissions, parameter mapping deviations, or incomplete coupling representations, driving solution deviation above 2%. Temperature therefore primarily affects the stability of the requirement parsing and modeling stages rather than the reliability of code execution.

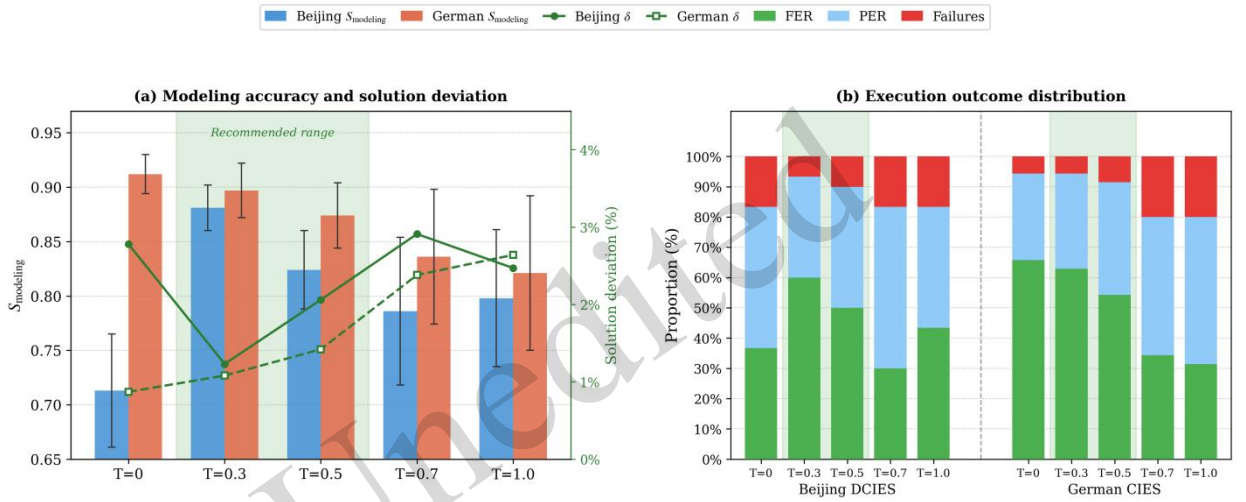

Fig. 13 Sensitivity of MASEO performance to the temperature parameter T. (a) Modeling accuracy and solution deviation; (b) execution outcome distribution

Examining the temperature ranges, T = 0 produces a clear divergence: the German CIES achieves its highest S_{modeling} (0.912), lowest solution deviation (0.87%), and narrowest error bands; the Beijing DCIES, by contrast, records its lowest S_{modeling} (0.713) and widest deviation (2.78%). This divergence is rooted in the difference in system complexity: an overly conservative generation strategy impairs information completeness in complex scenarios and incurs the highest token consumption (8.74 M for Beijing), yielding poor economic efficiency. In the range T ∈ [0.3, 0.5], both scenarios achieve good accuracy and stability-Beijing DCIES reaches S_{modeling} 0.881 with a failure rate of 7% at T = 0.3, while German CIES maintains 0.897-with token consumption at a reasonable level (7.05–8.03 M for Beijing), making this range a favorable balance between performance and cost. As T rises to 0.7 and above, error bands widen, failure rates climb to 17%–20%, and both modeling accuracy and solution deviation deteriorate; high-temperature sampling causes the PFA to terminate information confirmation prematurely, increasing constraint omissions. The reduction in token consumption at high temperatures (approximately 5.2 M for Beijing) stems from shorter dialogs rather than improved quality, offering limited practical benefit. Overall, T ∈ [0.3, 0.5] is the recommended working range.

Table 12 Token consumption and per-task cost across temperature settings

T	Beijing DCIES			German CIES		
	Input	Output	Cost ($)	Input	Output	Cost ($)
0.0	8.74	0.91	39.85	5.61	0.90	30.26
0.3	8.03	1.10	40.59	4.96	0.92	28.62
0.5	7.05	0.99	36.00	4.38	0.88	26.25
0.7	5.20	0.96	30.07	3.77	0.88	24.52

1.0	5.15	1.00	30.49	3.97	0.92	25.66

Acknowledgments

This work is supported by the National Natural Science Foundation of China (Grant No. 52576234), the National Key Research and Development Program of China (Grant No. 2023YFE0108600), the foundation of Key Laboratory of Cleaner Intelligent Control on Coal & Electricity, Ministry of Education, P. R. China (CICCE202510), and the Scientific Research Fund of Zhejiang University (XY20250460).

Author contributions

Boan QU: Conceptualization, Methodology, Software, Formal analysis, Visualization, Writing – original draft, Writing – review & editing. Haoyu JIANG: Visualization, Writing – original draft. Songjie WANG: Conceptualization. Zheng LUO: Writing – original draft. Xueru LIN, Xingtao TIAN, Jian LI, Xiaojie LIN, and Wei ZHONG: Supervision, Writing – review & editing.

Conflict of interest

Boan QU, Haoyu JIANG, Songjie WANG, Zheng LUO, Xueru LIN, Xingtao TIAN, Jian LI, Xiaojie LIN, and Wei ZHONG declare that they have no conflict of interest.

Data availability

The data that support the findings of this study are available from the corresponding author upon reasonable request.

References

Breyer C, Mohammadi-ivatloo B, Honkapuro S, et al., 2025. Overview of energy modeling requirements and tools for future smart energy systems. Renewable and Sustainable Energy Reviews, 212.

Di Bella A, Canti F, Prina MG, et al., 2024. Power system investment optimization to identify carbon neutrality scenarios for Italy. Environmental Research: Energy, 1(3):035001.

Diallo A, Bikakis A, Dickens L, et al., 2025. Rule-guided feedback: Enhancing reasoning by enforcing rule adherence in large language models. arXiv preprint, arXiv:2503.11336.

Du Y, Wei F, Zhang H, 2024. AnyTool: self-reflective, hierarchical agents for large-scale API calls. arXiv preprint arXiv:2402.04253.

Fan H, Wang CY, Liu L, et al., 2022. Review of uncertainty modeling for optimal operation of integrated energy system. Frontiers in Energy Research, 9:641337.

Gelaro R, McCarty W, Suárez MJ, et al., 2017. The modern-era retrospective analysis for research and applications, version 2 (MERRA-2). Journal of Climate, 30(14):5419-5454.

Hilpert S, Kaldemeyer C, Krien U, et al., 2018. The Open Energy Modelling Framework (oemof)-A new approach to facilitate open science in energy system modelling. Energy Strategy Reviews, 22:16-25.

Huang SJ, Lu H, Chen MZ, et al., 2023. Integrated energy system scheduling considering the correlation of uncertainties. Energy, 283:129011.

Jia MS, Cui ZY, Hug G, 2024. Enabling large language models to perform power system simulations with previously unseen tools: a case of daline. arXiv preprint arXiv:2406.17215.

Jiang HY, Qu BA, Zhu JJ, et al., 2025. HyperLoad: A Cross-Modality Enhanced Large Language Model-Based Framework for Green Data Center Cooling Load Prediction. arXiv preprint, arXiv:2512.19114.

Jiang HY, Zeng FJ, Qu BA, et al., 2025. Helios: A Foundational Language Model for Smart Energy Knowledge Reasoning and Application. arXiv preprint, arXiv:2512.19299.

Jiang JY, Wang F, Shen JS, et al., 2026. A survey on large language models for code generation. ACM Transactions on Software Engineering and Methodology, 35(2):1-72.

Jin HL, Chen HM, Lu QH, et al., 2025. Towards advancing code generation with large language models: A research roadmap. arXiv preprint, arXiv:2501.11354.

Jin M, Sel B, Hardeep F, et al., 2023. A human-on-the-loop optimization autoformalism approach for sustainability. arXiv preprint arXiv:2308.10380.

Li RD, Allal LB, Zi YT, et al., 2023. Starcoder: may the source be with you! arXiv preprint arXiv:2305.06161.

Lin XJ, Luo Z, Du-Ikonen L, et al., 2025. Generative artificial intelligence: Pioneering a new paradigm for research and education in smart energy systems. Energy and AI, 100610.

Lin XJ, Zhang N, Zhong W, et al., 2023. Regional integrated energy system long-term planning optimization based on multi-energy complementarity quantification. Journal of Building Engineering, 68:106046.

Lin Y, Lu ZM, Qin HQ, et al., 2026. Multi-Objective Optimal Dispatch of Integrated Energy Systems Guided by Dynamic Carbon Emission Factors. ACS Omega.

Liu M, Shi Y, Fang F, 2014. Combined cooling, heating and power systems: a survey. Renewable and Sustainable Energy Reviews, 35:1-22.

Liu SC, Chen CS, Qu XH, et al., 2024. Large language models as evolutionary optimizers. 2024 IEEE Congress on Evolutionary Computation (CEC), p.1-8.

Madaan A, Tandon N, Gupta P, et al., 2023. Self-refine: Iterative refinement with self-feedback. Advances in Neural Information Processing Systems, 36:46534-46594.

Michelon F, Zhou YH, Morstyn T, 2025. Large language model interface for home energy management systems. Proceedings of the 16th ACM International Conference on Future and Sustainable Energy Systems, p.590-602.

Priesmann J, Nolting L, Kockel C, et al., 2021. Time series of

useful energy consumption patterns for energy system modeling. Scientific Data, 8(1):148.
Roziere B, Gehring J, Gloeckle F, et al., 2023. Code llama: open foundation models for code. arXiv preprint arXiv:2308.12950.
Song D, Meng W, Dong M, et al., 2022. A critical survey of integrated energy system: summaries, methodologies and analysis. Energy Conversion and Management, 266:115863.
Team GLM, Zeng A, Xu B, et al., 2024. ChatGLM: a family of large language models from GLM-130B to GLM-4 all tools. arXiv preprint arXiv:2406.12793.
Tran KT, Dao D, Nguyen MD, et al., 2025. Multi-agent collaboration mechanisms: a survey of llms. arXiv preprint arXiv:2501.06322.
Wang S, Xiang D, et al., 2025. A multi-objective bilevel planning for data center integrated energy systems with waste heat utilization. Energy 335: 137934.
Wang YD, Wang JY, Chu ZW, 2025. Multi-agent large language models as evolutionary optimizers for scheduling optimization. Computers & Industrial Engineering:111197.
Wei J, Wang XZ, Schuurmans D, et al., 2022. Chain-of-thought prompting elicits reasoning in large language models. Advances in Neural Information Processing Systems, 35:24824-24837.
Yang CR, Wang XZ, Lu YF, et al., 2023. Large language models as optimizers. The Twelfth International Conference on Learning Representations.
Yang YF, Zhou ZC, Xiao XB, et al., 2024. Intelligent day-ahead optimization scheduling for multi-energy systems. Frontiers in Energy Research, 11:1349194.
Zhu HH, Wang XL, Wen YT, et al., 2025. A review of integrated energy system modeling and operation. Applied Energy, 400:126572.

Electronic supplementary materials

Sections S1-S4, Tables S1-S11, Algorithm S1, Figs. S1-S6

中文概要

题　目：面向综合能源系统调度的多智能体协同优化框架

作　者：曲泊安[1]，江浩宇[2]，王松杰[2]，罗政[2]，林雪茹[2,3]，田兴涛[4]，李健[5]，林小杰[2,6,7]，钟崴[1,2]

机　构：[1]浙江大学，工程师学院，中国杭州，310027；[2]浙江大学，能源工程学院，中国杭州，310027；[3]浙大城市学院，信息与电气工程学院，中国杭州，310015；[4]太原理工大学，煤电清洁智能控制教育部重点实验室，中国太原，030024；[5]北京理工大学，机械工程学院，中国北京，100081；[6]浙江大学，上海高等研究院，中国上海，200120；[7]浙江大学，常州工业技术研究院，中国常州，213000

目　的：综合能源系统（IES）调度优化涉及多能耦合、多设备协调与复杂约束，传统方法依赖领域专家手工构建优化模型，存在技术门槛高、开发周期长、可迁移性差等问题。本文旨在研究基于大语言模型的多智能体协同框架，以降低 IES 调度优化对专业建模专长的依赖，提升端到端自动化程度与求解可靠性综合能源系统（IES）调度优化涉及多能耦合、多设备协调与复杂约束，传统方法依赖领域专家手工构建优化模型，存在技术门槛高、开发周期长、可迁移性差等问题。本文旨在研究基于大语言模型的多智能体协同框架，以降低 IES 调度优化对专业建模专长的依赖，提升端到端自动化程度与求解可靠性。。

创新点：1. 提出 MASEO 多智能体协同框架，将调度优化流程分解为信息采集、数学建模、代码生成与执行验证四个阶段，由四个专用智能体串行协作完成；2. 设计结构化检查表引导信息收集，结合基于错误分类的可追溯修复机制，将执行失败精准路由至责任上游智能体进行定向修正。

方　法：1. 通过结构化检查表与多轮人机对话，由问题建模智能体（PFA）完整采集系统信息（图 3、4）；2. 能量建模智能体（EMA）基于检查表与历史模型检索结果构建数学优化模型，代码生成智能体（CGA）结合 RAG 框架选择匹配求解框架并生成可执行代码（公式(3)、(4)）；3. 执行反馈智能体（EFA）执行代码并进行物理可行性验证，依据错误分类表路由修复（表 1），最大自动修复迭代次数为 3 次；4. 以北京数据中心 IES 和德国社区 IES 为案例，通过对比实验、消融实验、多骨干模型对比及温度敏感性分析对框架进行系统评估（图 11–13）。

结　论：1. MASEO 框架可将原本需数天至数周的 IES 调度优化压缩至数小时内完成，北京案例年化总成本偏差约 1.4%，德国案例结果与专家基准一致；2. 与单智能体基线相比，建模准确率提升约 35%，代码执行通过率从 63%提升至 94%；消融实验表明结构化检查表、可追溯修复机制与 RAG 检索三个模块分别独立作用于建模稳定性、执行可靠性与代码质量；3. 温度参数 $T \in [0.3, 0.5]$为推荐工作区间；DeepSeek-V3 与 Qwen3-32B 在性能与成本之间具有良好平衡，适合资源受限场景部署。

关键词：综合能源系统；优化调度；大语言模型；多智能体系统