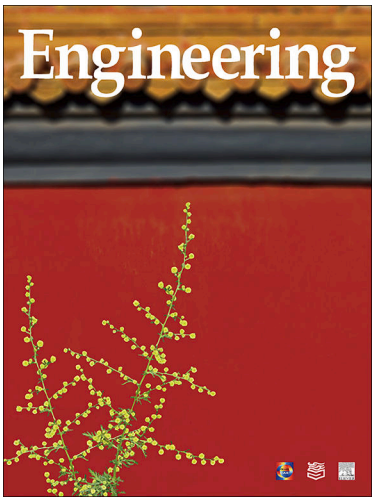
Journal Pre-proofs



Article

LLM-Driven Framework for Industrial Design Automation

Sicheng He, Xiaoxu Wang, Zeke Chen, Jianxing Liao, Bo Wang, Junyan Xu, Xiaohong Guan, Shui Yu, Yun Li

PII: S2095-8099(26)00201-8
DOI: https://doi.org/10.1016/j.eng.2026.04.009
Reference: ENG 2324

To appear in: *Engineering*

Received Date: 2 June 2025
Revised Date: 12 December 2025
Accepted Date: 9 April 2026

Please cite this article as: S. He, X. Wang, Z. Chen, J. Liao, B. Wang, J. Xu, X. Guan, S. Yu, Y. Li, LLM-Driven Framework for Industrial Design Automation, *Engineering* (2026), doi: https://doi.org/10.1016/j.eng.2026.04.009

Research

Applications of ChatGPT—Article

LLM-Driven Framework for Industrial Design Automation

Sicheng He[a,b], Xiaoxu Wang[a,b], Zeke Chen[a,b], Jianxing Liao[a,b], Bo Wang[a,b], Junyan Xu[a,b], Xiaohong Guan[c], Shui Yu[a,b*], Yun Li[a,b,c] *

[a] *SIAS, University of Electronic Science and Technology of China, Chengdu 611731, China*

[b] *Shenzhen Institute for Advanced Study, UESTC, Shenzhen 518000, China*

[c] *i4AI Ltd, London WCIN 3AX, UK*

* Corresponding authors,
Email: Shui.Yu@i4ai.org, Yun.Li@ieee.org

Abstract

The rapid advances of large language models (LLMs) have presented industrial sectors with transformative opportunities for innovative designs beyond the capabilities of human designers. Due to the inherent black-box nature of LLMs, however, existing LLM-based design frameworks lack the explainability and robustness that human designers would otherwise offer. To address this issue, we propose an LLM-driven Industrial Design Automation (LLM-IDA) framework to encompass conceptual design, knowledge-based design, and digital prototyping. The LLM-IDA utilizes a multi-tiered artificial intelligence (AI) agent group for four modular processes: ① a specification module, ② a quantification module, ③ a surrogate module, and ④ a prototype module. With every module autonomously executed by the agent group, LLM-IDA enhances the generation of novel designs with few-shot prompts and completes the design process with minimal human intervention. To validate this method, the LLM-IDA is tested and compared with a state-of-the-art retrieval augmented generation (RAG) method using the pass@10 metric. Ablation tests confirm the positive impact of each module on both time costs and optimization performance. Overall, the experimental results show that the LLM-IDA delivers performance superior to the RAG method in real-world applications.

Keywords:

Computer-aided design, Large language models, Industrial design automation, Multi-agent systems, Design optimization

1. Introduction

Industrial design, situated at the intersection of creativity, engineering, and user-centric innovation, aims to transform abstract concepts into functional, manufacturable products through systematic processes. A typical industrial design workflow involves market research, requirement analysis, conceptual design, detailed design, and prototyping, with each phase requiring iterative collaboration between human expertise and technical tools. However, traditional design optimization often depends on substantial human intervention, which increases design costs and slows the advancement of design and manufacturing towards greater creativity, lower costs, and better responsiveness to customer needs. Computer-Automated Design (CAutoD) seeks to achieve fully automated design by leveraging the interactions among different systems and design tools to deliver customized design [1,2]. We identify two main challenges for CAutoD: ① encoding human design knowledge into automated systems and ② enabling such systems to comprehend and interact with the physical world for product evaluation and optimization.

The emergence of large language models (LLMs) offers significant potential to address these challenges. Owing to their advanced natural language processing capabilities, LLMs demonstrate remarkable proficiency in understanding contextual nuances and generating human-like responses. Recent studies have highlighted their applicability across various stages of industrial design. In market research, LLMs that are fine-tuned using survey data align closely with human responses to product features [3]. For requirement engineering, Arora et al. [4] proposed LLM-enhanced frameworks to improve task accuracy. In conceptual design, AI-MCD (AI-augmented Multimodal Collaborative Design) frameworks utilize iterative LLM prompting to generate visual schemes, whereas morphology analysis methods refine design decomposition and synthesis [5]. Despite these advances, the detailed design phase—where computer-aided design (CAD) and computer-aided engineering (CAE) tools translate concepts into producible specifications—still faces persistent bottlenecks. LLM hallucinations [6] require strategies such as fine-

tuning to ensure reliable tool invocation, as demonstrated by Jiang et al. [7] for application programming interface (API)-driven automation.

To address the aforementioned challenges of automation and reliability, this paper introduces the LLM-driven Industrial Design Automation (LLM-IDA) framework, integrating three levels of AI-Agent (Fig. 1).

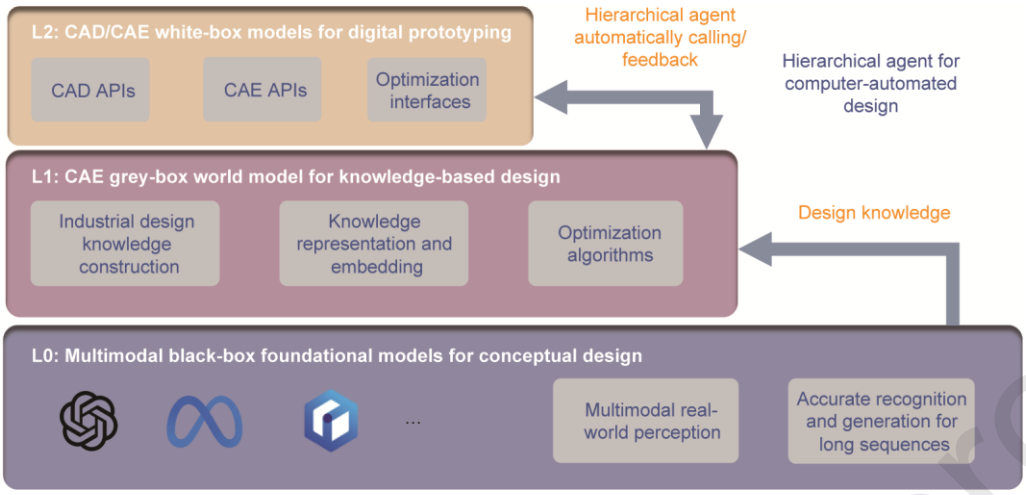


Fig. 1. AI-agent-based CAutoD layers driven by an LLM, CAE knowledge, and CAD prototyping: L0 for conceptual design, L1 for knowledge-based design, and L2 for digital prototyping through optimization

L0 supports conceptual design and requirement analysis, where intelligent agents utilizing multimodal black-box foundational models interact with users to ascertain precise requirements. This layer leverages the multimodal capabilities of LLMs and foundational domain knowledge to conduct an initial conceptual design, which is then passed to L1. L1 is responsible for knowledge-based design. In this layer, agents apply constraints derived from first principles by integrating a knowledge base and a knowledge graph containing specialized CAE expertise. By incorporating professional knowledge, L1 evolves the initial concept into a product that broadly meets current needs, thereby narrowing the search space for subsequent simulation optimization. Specifically, in the context of multi-agent systems [47], L1 agents enhance the design by leveraging LLMs to guide the conversion of natural language descriptions into more refined design elements. These agents define specific constraints and ensure that the design satisfies both functional and manufacturing requirements. These agents also integrate multimodal feedback from multiple sources, thereby improving design fidelity. Layer L2 focuses on digital prototyping and detailed design using white-box knowledge within CAD software. It utilizes the code-generation capabilities of LLMs to invoke CAD/CAE tools, first transforming the conceptual design into a CAD model. Subsequently, the CAE software APIs perform finite element analysis (FEA) on this model and optimize product performance based on the simulation results.

Although previous studies demonstrated substantial advances in automating conceptual design through iterative prompting and multimodal collaboration mechanisms, significant challenges remain for knowledge-based design and digital prototyping. To address these limitations, we propose a threefold strategy: ① extracting inherent commonsense knowledge from LLMs to guide decision-making, ② integrating domain-specific design knowledge to refine system outputs, and ③ modularizing LLM agents to mimic human designers' role-specific workflows, thereby enabling precise task execution in tool invocation, physical simulation, and iterative optimization. In this study, we focused on validating the feasibility of LLM-IDA through industrial design optimization, a scenario that inherently bridges knowledge-based design (e.g., applying engineering principles) and digital prototyping (e.g., simulating physical behaviors). A multi-tiered artificial intelligence (AI) agent group is used for LLM-based code generation to connect optimization algorithms with design tools to enhance existing design automation schemes. In LLM-IDA-based design optimization, the agent group acts as the core controller, analyzing the optimization problems in the LLM using the Chain of Thought (CoT) [8] based on pretrained knowledge and the provided knowledge base. It then utilizes retrieval-augmented generation [9] (RAG) to call existing tools (including simulation and optimization tools).

Overall, our contributions are as follows: ① We propose the LLM-IDA solution for industrial design automation; ② for the first time, high-level automation of design optimization is achieved; ③ experimental results show that the defined code templates not only enhance the stability of the generation but also possess a degree of generality; ④ by automatically calling simulation software APIs, the LLM-IDA provides feedback on complex physical behavior, eliminating reliance on the LLMs' mathematical capabilities in the optimization process; ⑤ before optimization, the LLM-IDA automatically establishes a surrogate model

through multiple rounds of dialogue, reducing time-consuming repetitive simulations; and ⑥ LLMs are employed multiple times within the LLM-IDA to inspect and provide feedback on execution results, enhancing the reliability of the automated framework.

2. Related work

2.1. Large language model

Introduced in 2017, the transformer architecture [10] revolutionized the processing of sequential data through self-attention and laid the groundwork for subsequent language models. In 2018, the release of the BERT model [11] further enhanced language understanding capabilities, marking a significant milestone in natural language comprehension through pre-training with deep bidirectional transformer encoders. Subsequently, OpenAI's GPT series [12–14] demonstrated the potential of LLMs across multiple domains with vast parameter sets and text generation capabilities. Meanwhile, the advent of multimodal models such as DALL-E [15] and Stable Diffusion [16] has expanded the application of language models into the visual domain by not only processing text but also understanding and generating images. Large language and multimodal models have flourished, profoundly transforming human production and everyday life.

The application of LLMs in vertical domain has demonstrated significant potential. Through fine-tuning techniques, such as LoRA [17], adapter tuning [18], and prefix tuning [19], large models can be adapted to datasets specific to certain tasks or domains, thereby enhancing their performance in targeted scenarios. Moreover, prompt engineering allows users to guide models to generate specific outputs by designing input prompts or instructions, thereby improving the model performance in specialized tasks without requiring fine-tuning. Additionally, the application of multi-agent systems [20] enables large models to engage in more complex and dynamic interactions, providing more efficient and intelligent solutions across industries such as finance, healthcare, and manufacturing.

In addition to these advancements, a suite of innovative techniques are propelling the evolution of large-scale models. Among these, RAG, introduced by Facebook AI Research, seeks to harmonize retrieval mechanisms with generative models to mitigate the issues of knowledge decay and hallucinations that plague generative models when handling extensive texts. At its heart, RAG operates by fetching pertinent information from external knowledge repositories during the text-generation process and seamlessly incorporating it into the generative framework.

Mathematically, the generation process in RAG can be encapsulated by the formula

$$P(y|x) = \sum_{z \in Z} P(y|x,z)P(z|x) \tag{1}$$

where x is the input text, y is the generated text, Z is the set of relevant information retrieved from the external knowledge source, $P(z|x)$ represents the probability of retrieving information z given input x, and $P(y|x,z)$ represents the probability of generating text y given input x and retrieved information z. Accordingly, CoT is a methodology that encourages language models to produce a sequence of intermediate reasoning steps, thereby steering the model toward the final answer. Furnishing the model with examples that include chains of thought makes the model more adept at tackling tasks requiring multistep reasoning. For instance, in the context of a mathematical problem, CoT can guide the model to unfold the reasoning process methodically, rather than delivering the answer outright.

Mathematically, CoT can be regarded as a sequence of reasoning steps $(s_1, s_2, \cdots, s_n)$, through which the answer a is derived from the question q, expressed as

$$q \overset{s_1}{\rightarrow} s_2 \overset{s_2}{\rightarrow} \ldots \overset{s_n}{\rightarrow} a. \tag{2}$$

Ning et al. [21] proposed that the reasoning process of language models is similar to probabilistic pattern matching on Monte Carlo language trees (such as GPT-Tree and Data-Tree). For CoT, when facing complex problems, the input is at a certain parent node of the GPT-Tree, and the answer is at a deeper leaf node, making direct derivation more difficult. CoT enables the model to derive answers step by step by generating intermediate reasoning that bridges the gap between input and output.

The prospects for LLMs in the industrial sector are exceedingly broad. As technology advances and the market matures, large models gradually transition from general-purpose scenarios to high-value vertical domains. Large industrial models have already shown preliminary applications in industries such as semiconductors [22] and mechanical design [23]. In the future, with further integration into corporate systems, these models could form a model layer that interfaces with various internal data and business systems while offering APIs or directly serving as unified business entry points, thereby accelerating the process of industrial automation and intelligence.

2.2. Industrial design optimization

The design cycle of industrial products typically comprises several phases, including market research, requirements analysis, conceptual design, and detailed design. In engineering and industrial design, design optimization is a systematic approach involving conceptual design, knowledge-based design, and digital prototyping. The latter two processes involve repetitive

simulations to determine the optimal combination of design variables under given constraints to achieve specific objectives, such as minimizing costs, maximizing efficiency, or enhancing performance.

The implementation of design optimization relies on efficient algorithms and computational tools. To address complex optimization problems, researchers have developed various heuristic and metaheuristic algorithms that mimic natural phenomena to guide the search for better solutions. Modern design optimization methods incorporate advanced algorithms, such as particle swarm optimization (PSO) [24], genetic algorithms (GA) [25], simulated annealing (SA) [26], and differential evolution (DE) [27], as well as their application in various engineering fields, such as structural design [28–32] and circuit design [33–36]. Through algorithm-assisted decision making, engineers can achieve significant improvements in product performance and maximize cost efficiency.

It is worth noting that computational workload is a major challenge in many engineering and industrial optimization problems because the most time-consuming part of the optimization process is often the evaluation of the objective function [37,38]. In addition to using formulas derived through mathematical deduction or FEA available in industrial software, another approach is to employ surrogate models. This method can be more efficient if the number of high-fidelity model evaluations is significantly reduced [39–41]. Thus, the combination of low- and high-fidelity models in surrogate-based optimization can be a powerful tool for many practical applications.

2.3. LLMs for industrial design

The development of LLMs presents significant opportunities for the advancement of industrial designs. In the realm of market research, Horton [42] demonstrate that various LLMs respond to classic behavioral economics experiments in ways that align with intuition and experience. Brand et al. [3] introduced a practical approach for market researchers to enhance the GPT responses by fine-tuning them using survey data from similar contexts. This method improves the consistency of GPT responses with human responses regarding existing, and more importantly, new-product features. Furthermore, in requirement analysis, Arora et al. [4] explored the potential of LLMs in advancing the Requirements Engineering (RE) process, proposing key directions and a strengths–weaknesses–opportunities–threats (SWOT) analysis for research and development using LLMs, thereby enhancing the efficiency and accuracy of demand-related tasks.

Xu et al. [5] proposed an AI-enhanced multimodal collaborative design framework that leveraged LLMs to establish iterative prompting mechanisms to generate precise visual schemes. Chen et al. [43] introduced a morphology analysis method for conceptual design enhanced by LLMs, which provides designers with targeted guidance and support by refining the design process into three main stages: decomposition, generation, and combination, thereby offering targeted guidance and support to designers when applying morphological analysis.

In the realm of detailed design, it is necessary to use CAD or CAE tools to transform conceptual designs into tangible and producible products. However, owing to the hallucination issues in LLMs [6] that can lead to errors in tool invocation, it is essential to employ fine-tuning or RAG for these models. Jiang et al. [7] facilitated the automation of detailed designs by fine-tuning LLMs to optimize API calls using industrial software. Qin et al. [44] achieved optimized design for shear wall structures by translating CAE engineers' simulation intentions into code. Similarly, this study primarily addresses automation challenges in detailed design, along with a concise analysis of business-to-business (B2B) requirements.

3. LLM-leveraged industrial design automation framework

In industrial design optimization, optimization algorithms, simulation software, and surrogate models are closely integrated, collectively driving the continual refinement and enhancement of design solutions. As the central driving force, optimization algorithms navigate the design space using specific mathematical strategies to seek optimal solutions, thereby providing a direction and methodology for design improvements. Simulation software serves as a vital analytical tool capable of replicating various physical phenomena and performance metrics of the design model, enabling precise evaluation of its behavior under real-world conditions and supplying essential feedback to the optimization algorithms. Acting as a bridge, surrogate models utilize data generated by simulation software to establish rapid approximations between design variables and performance outcomes through modeling techniques, effectively replacing complex and time-consuming simulations. This significantly boosts the efficiency of the optimization search and empowers engineers to achieve superior industrial design outcomes within limited timeframes and resource constraints.

Following the analysis of industrial design optimization, the automated workflow (as illustrated in Fig. 2) is divided into two principal modules: a specification module and a code-generation module; the latter is further segmented into three sub-modules: a quantification module for CAE simulation, a surrogate module for modeling, and a prototyping module for optimization. The primary function of the specification module is to allocate tasks to each submodule within the code-generation module and orchestrate the interfaces among the optimization algorithms, simulation software, and surrogate models to accomplish a fully automated design optimization process.

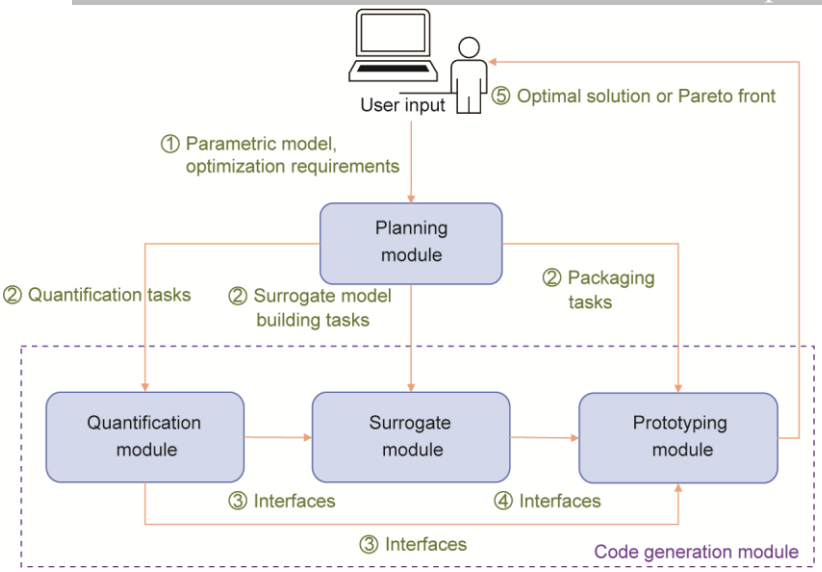


Fig. 2. Overall framework of the LLM-IDA based design optimization: The specification module dissects and allocates tasks, whereas the code-generation module interlinks simulation, surrogate model construction, and optimization through code generation and interface encapsulation.

However, the implementation of the three code-generation modules using LLMs faces fundamental probabilistic challenges. From a set-theoretic perspective, let Ω denote the infinite problem space of FEA pre-processing, where each element $\omega \in \Omega$ represents a unique combination of geometry, loads, and boundary conditions. In contrast, the external knowledge base used in RAG is a finite set K, where $|K| \ll |\Omega|$. The probability of retrieving a valid reference $\kappa \in K$ that falls within the semantic neighborhood ϵ of a novel query ω_{new} is governed by

$$P(\text{Success}_{\text{RAG}}) \approx P(\exists \kappa \in K: \|\omega_{\text{new}} - \kappa\| < \epsilon). \tag{3}$$

As the dimensionality of the boundary conditions $D \to \infty$, the density of K in Ω approaches zero ($\lim_{D\to\infty} \frac{|K|}{|\Omega_D|} = 0$), causing the retrieval confidence to collapse and forcing the model to hallucinate based on weak priors.

We put forward a "Task Analysis–Code Generation–Code Feedback" framework to address this challenge, which converts the generation process into a conditional probability chain. First, the Task Analysis module acts as a surjective mapping function $f: \Omega \to S$, projecting the high-dimensional natural language input onto a structured low-dimensional intent space S. This significantly reduces the information entropy, where $H(\cdot)$ denotes the Shannon entropy of a given space. Consequently,

$$H(S) \ll H(\Omega). \tag{4}$$

By leveraging this structured intent S, the code-generation module approximates the distribution $P(C|S)$, where C denotes the generated code candidate. To rigorously quantify the reliability, we model the validation process as a sequential probabilistic chain involving a posterior discriminator E. Let $p_i = P(C_{\text{valid}}|S, \epsilon_{i-1})$ denote the success probability at the i-th iteration, where ϵ_{i-1} represents the execution error feedback from the $(i-1)$-th attempt. The system only fails if all N consecutive attempts fail. Assuming conditional independence given the feedback history, the joint probability of total failure is the product of individual failure probabilities: $P_{\text{fail}}^{(N)} = \prod_{i=1}^{N}(1 - p_i)$. Consequently, the cumulative probability of generating a valid code converges to complement the total failure probability.

$$P\left(C_{\text{valid}}|S, E\right) = 1 - P_{\text{fail}}^{(N)} = 1 - \prod_{i=1}^{N}\left[1 - P\left(C_{\text{valid}}|S, \epsilon_{i-1}\right)\right]. \tag{5}$$

As N increases, the failure term approaches zero, thereby ensuring robustness. Moreover, to illustrate the workflow of the LLM-IDA framework concretely, we employed the automated optimization design process of a three-section cantilever beam as a case study, as shown in Fig. 3. This setup involved fixing the left side of the beam while applying a force to the right side. The objective of the optimization was to develop a model that achieved a balance between minimal volume and high deformation resistance. This multi-objective optimization problem is mathematically formulated in Eq. (6), where the decision variable vectors $\boldsymbol{x} = [l_1, l_2]^{\mathrm{T}}$ represent the lengths of the first two sections, and l_i denotes the length of the i-th section. The objective function simultaneously minimizes $\text{Volume}(\boldsymbol{x})$ (in m^3) and $\text{MaxDisplacement}(\boldsymbol{x})$ (in m). Subject to geometric constraints $0 < l_1, l_2 < 1$. Specifically, to unify the compilation environment and facilitate the implementation of a fully automated process, we employed

the PyAnsys library to invoke the Ansys kernel for the simulation, meaning that the three-dimensional (3D) models are also parametrically generated via the PyAnsys modeling code.

$$\begin{aligned} &\min && \{f_1(\boldsymbol{x}), f_2(\boldsymbol{x})\} = \{\text{Volume}(\mathbf{x}), \text{MaxDisplacement}(\boldsymbol{x})\} \\ &\text{s.t.} && \boldsymbol{x} = [l_1, l_2]^{\mathrm{T}} \\ & && 0 < l_i < 1, i = 1{,}2 \end{aligned} \tag{6}$$

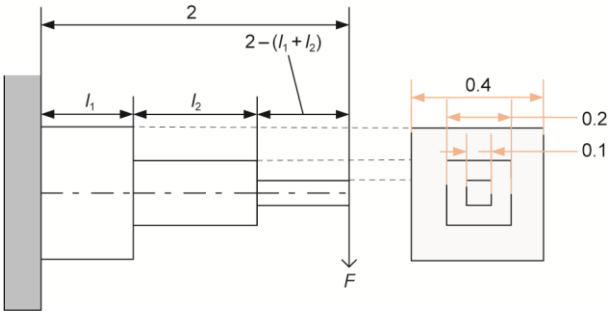


Fig. 3. Three-section cantilever beam example. F: force.

3.1. Specification module

To bridge the gap between the abstract user intent and executable engineering codes, we formally defined the automated design process as a mapping from the natural language space ($\mathcal{L}$) to the optimization problem space ($\mathcal{O}$). The specification module functions as the primary mapping operator Φ. Given a natural language request $R_{\mathrm{NL}} \in \mathcal{L}$, the specification module parses it into a standardized optimization tuple:

$$\Phi(R_{\mathrm{NL}}) \to \mathcal{P}_{\mathrm{opt}} = \{\boldsymbol{x}, \Omega, f, \mathcal{C}\} \tag{7}$$

where the resulting optimization problem is defined as

$$\begin{aligned} &\underset{\boldsymbol{x}}{\text{minimize}} && \left(f_1(\boldsymbol{x}), f_2(\boldsymbol{x}), \ldots, f_s(\boldsymbol{x})\right) \\ &\text{subject to} && g_i(x) \le 0, \quad i = 1{,}2, \ldots, m \\ & && h_j(x) = 0, \quad j = 1{,}2, \ldots, p \\ & && x \in \Omega \subseteq \mathcal{R}^n \end{aligned} \tag{8}$$

where n denotes the dimensionality of the decision variable vector x, m and p represent the numbers of inequality and equality constraints, respectively. The functions $f_s(x)$ denote the objective functions to be minimized, $g_i(x)$ and $h_j(x)$ denote the inequality and equality constraint functions, respectively.

Drawing on this formulation, the interaction between the specification module and subsequent code-generation modules is mathematically formalized as follows:

(1) Design variables and solution space ($\boldsymbol{x}, \Omega$): The specification module extracts the design variables $\boldsymbol{x} = [x_1, x_2, \ldots, x_n]^{\mathrm{T}}$ and their bounds from R_{NL}, which defines the search space. The tuple ($\boldsymbol{x}, \Omega$) serves as the input to the surrogate module, determining the dimensionality of the sampling strategy.

(2) Objective function (f): Objective $f(\boldsymbol{x})$ represents the performance metric to be optimized (e.g., mass and stress). The specification module identifies whether $f(\boldsymbol{x})$ is an analytical equation or a black box simulation API. This definition is passed on to the quantification module to generate the corresponding code.

(3) Constraint ($\mathcal{C}$): The set of constraints $\mathcal{C} = \{g_i, h_j\}$ defines a feasible region. These constraints, along with the optimization strategy, are passed to the prototyping module to formulate the final solver script.

By establishing these formal definitions, the specification module ensures that the "abstract-to-concrete" conversion is mathematically consistent across all sub-modules.

Fig. 4 shows an example workflow of the specification module. From this, we can infer that the analyzer initially evaluates the shape and size of the model using natural language, identifies the optimization variables and solution space, and ultimately determines the number and nature of optimization objectives. The results demonstrate that the analyzer can accurately identify the shape of the model, parametric model variables, solution space, and optimization objectives. Based on this analysis, the task allocator further decomposes the tasks of the coder modules. The objective-quantification context provides a method for volume calculations and API call requirements. For the surrogate module, the input and output of the neural network are defined precisely.

For the packaging module, the solution space of the two variables was accurately provided, and the interfaces for the two objective functions were defined to maintain uniformity in the interface calls. Additionally, the allocator determines the algorithm used. By reviewing the workflow of the specification module, we can infer that the system, which consists of an LLM, can flawlessly analyze this optimization problem without human intervention.

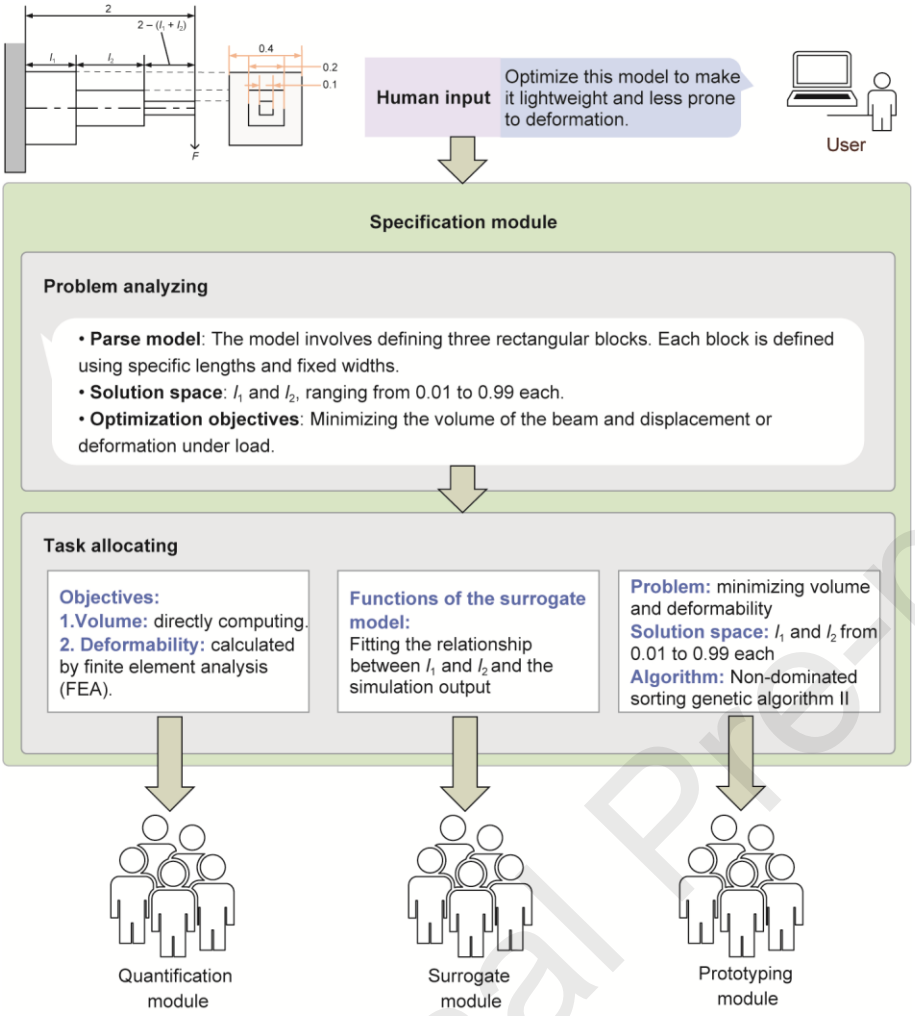


Fig. 4. Dialogue in the specification module, the allocation of design optimization tasks is collaboratively accomplished by the problem analyzing agent and task allocation agent.

3.2. Code-generating module

The code-generating module comprises three distinct submodules: quantification, surrogate, and prototyping. We illustrate the workflow of these modules using the three-section cantilever-beam case study introduced in Section 3.1.

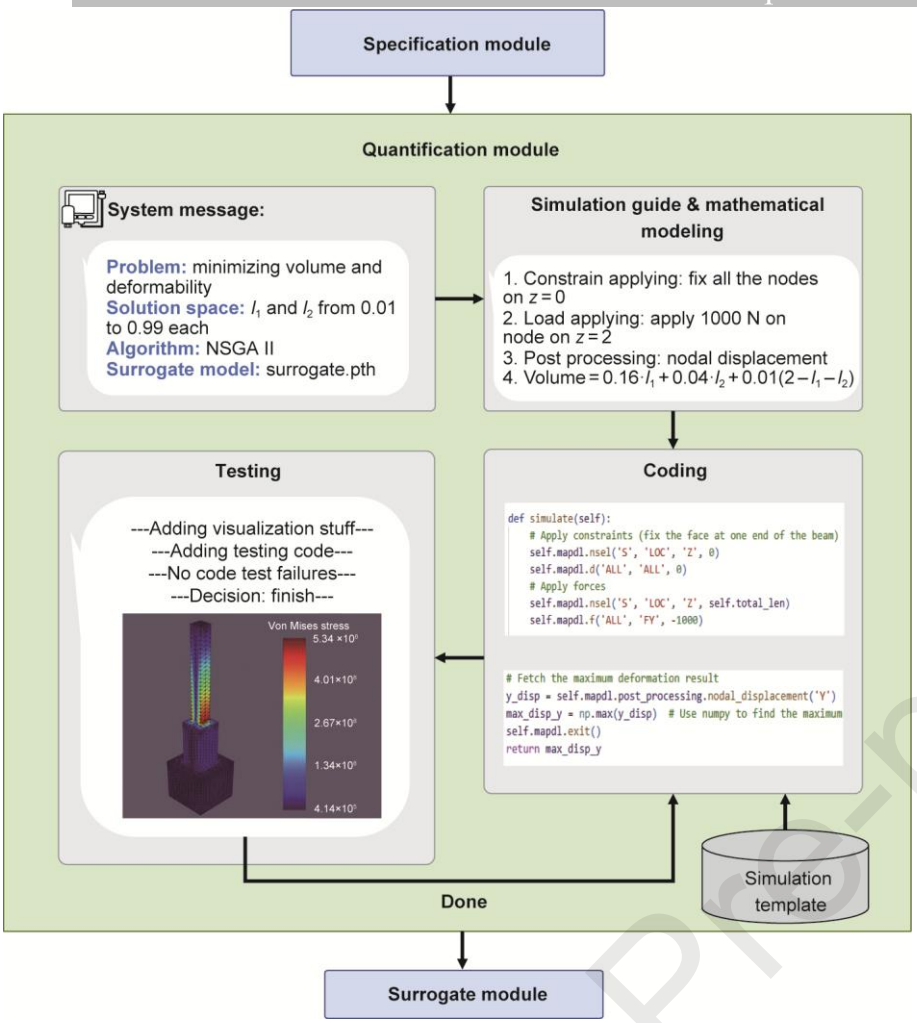


Fig. 5. Workflow of the quantification module using the three-section cantilever beam case. The module translates tasks into executable PyAnsys scripts through an iterative coding-testing loop.

Fig. 5 illustrates that the quantification module acts as a bridge between abstract specifications and executable simulations. Based on the task analysis from the specification module, the workflow initiates with the System Message and Simulation Guide, which decompose the physical problem into actionable steps (e.g., "① Calculate coordinate of constraint", "② Calculate coordinate of load"). Algorithm 1 describes the iterative process. Taking the CAD model (m) and quantification requirements (qr) as inputs, the LLM first analyzes the problem before entering a code-generation loop. This loop ensures that the generated simulation interface (I_{simu}) is not only syntactically correct (checked by $exeSucceed$) but is also logically valid based on the simulation results.

Algorithm 1: Objective quantification and interface generation

Input: CAD model m, quantification requirement $qr_1, qr_2, \ldots, qr_n$, Pyansys API document a, CAE template t, problem analyze prompt $\mathcal{P}_1$, code generation prompt $\mathcal{P}_2$, execution prompt $\mathcal{P}_3$

Output: Simulation/calculation interface I_{simu}

Iteration count initialization: $iteration \leftarrow 0$;

Decision variable initialization: $pass \leftarrow False$;

for $i = 1$ to n **do**

Problem Analyze: ProblemAnalyze$_i \leftarrow \text{llm}(m, qr_i, \mathcal{P}_1)$;
while NOT pass AND iteration < 5 **do**
Iteration count setting: iteration $\leftarrow$ iteration $+ 1$
Determination variable initialization: $exeSucceed \leftarrow$ False;
Error message initialization: error $\leftarrow$ None;
while NOT exeSucceed **do**
Code generation: $I_{\text{simu}} \leftarrow \text{llm}(\text{ProblemAnalyze}_i, a, t, \mathcal{P}_2, \text{error})$;
Code execution: simuRes, error $\leftarrow \text{Execute}(I_{\text{simu}})$;
If error $=$ None **then**
Simulation result validation: $pass \leftarrow \text{llm}(simuRes, \mathcal{P}_3)$;
Execution determination variable Assignment: exeSucceed $\leftarrow$ True;
if pass **then**
Return res;
end if
end if
end while
end while
end for

Specifically, for the cantilever beam, the coding agent utilizes the PyAnsys library to translate these steps into script commands, such as mapdl.nsel for node selection and mapdl.f for load applications. Importantly, this process includes an iterative testing loop. The Tester agent executes the generated script to produce a 3D visualization (e.g., von Mises stress contours). This verifies whether the boundary conditions and loads were applied correctly (e.g., ensuring that the fixed support is at $z = 0$ and the force is at $z = 2$). Once the visualization passes the "visual inspection," the module outputs the verified simulation script.

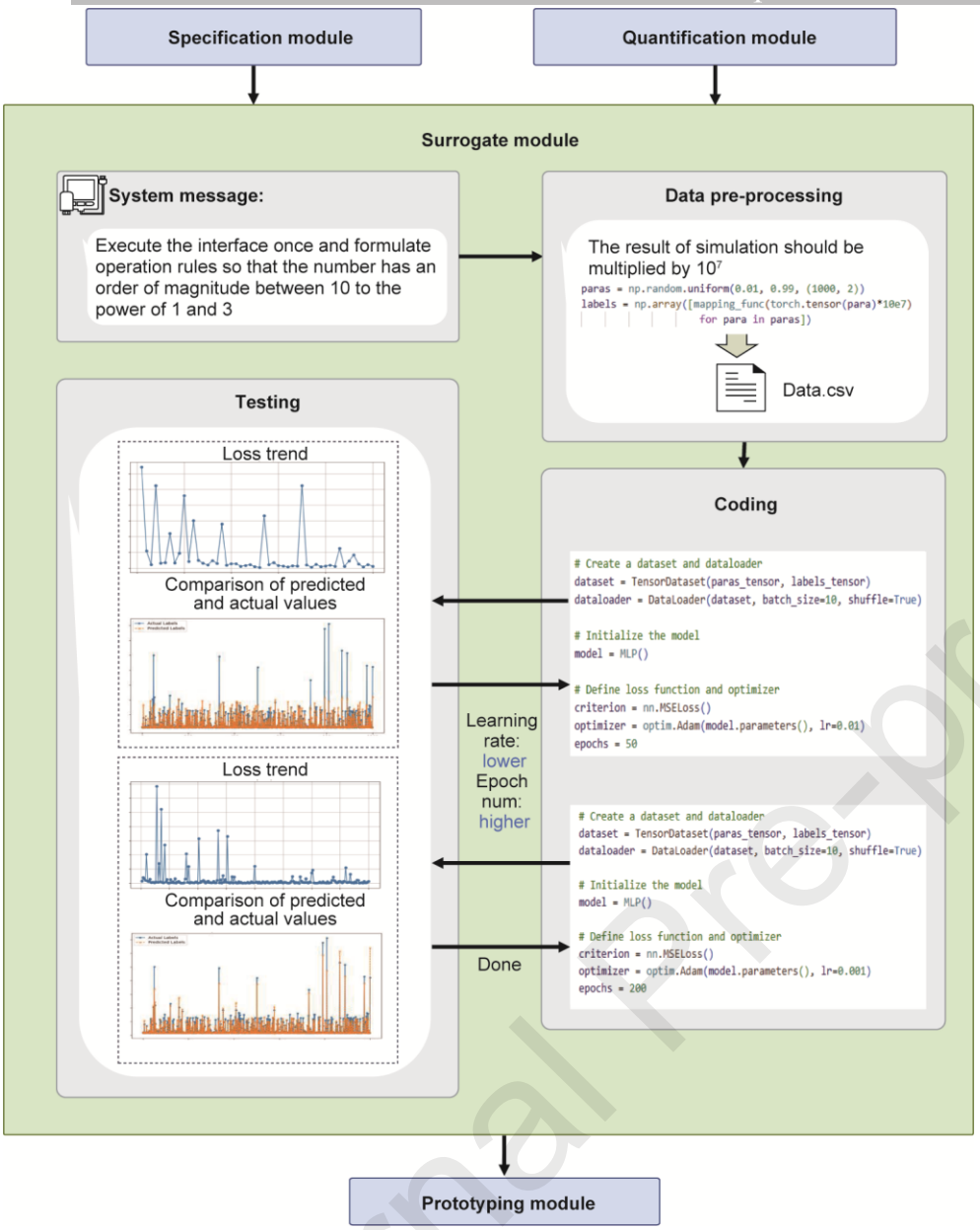


Fig. 6. Workflow of the surrogate module. The system performs uniform sampling, adaptive normalization (e.g., scaling by 10^7), and trains a neural network to approximate the FEA solver.

In real-world engineering, direct FEA is often computationally expensive. To accelerate the optimization, we employ a surrogate module, as depicted in Fig. 6. Algorithm 2 provides the procedural details of this module, starting with data pre-processing using the simulation interface generated in the previous step. A key distinction in this algorithm is that the LLM generates both the surrogate model structure ($I_{\text{surrogate}}$) and associated hyperparameters (HP). The process begins with data Pre-processor, which prompts the LLM to perform uniform sampling within the solution space to generate a few-shot dataset. Adaptive normalization is a critical feature, as detailed in the preprocessor block, that automatically detects the magnitude differences between objectives and scales them (e.g., multiplying by 10^7) to prevent numerical dominance. Finally, the Testing block evaluates the accuracy of the model and utilizes a dynamic feedback mechanism to automatically retrain the network with adjusted hyperparameters (e.g., learning rate and epochs) if the convergence criteria are not satisfied.

Algorithm 2: neural network surrogate model construction

Input： fitting task T, simulation interface I_{simu}, hyperparameters HP, data preprocess prompt $\mathcal{P}_1$, code generation prompt $\mathcal{P}_2$, execution prompt $\mathcal{P}_3$

Output： surrogate model interface $I_{\text{surrogate}}$

Iteration count initialization: iteration ← 0;
Decision variable initialization: pass ← False;
Data preprocess: Data ← llm$(T, \mathcal{P}_1, I)$;
for $i = 1$ to n **do**
  while NOT pass AND iteration < 5 **do**
    Iteration count setting: $iteration \leftarrow$ iteration $+ 1$
    Error message initialization: $error \leftarrow$ None;
    Code generation: $I_{\text{surrogate}}, H \leftarrow$ llm$(Data, \mathcal{P}_2, advice)$;
    Code execution: M, error, convergenceCurve ← Execute$(I_{\text{surrogate}}, HP)$;
    If error $=$ None **then**
      Result validation: pass, advice ← llm(error, convergenceCurve, $\mathcal{P}_3$);
      if pass **then**
        Return $I_{\text{surrogate}}$;
      end if
    end if
  end while
end for

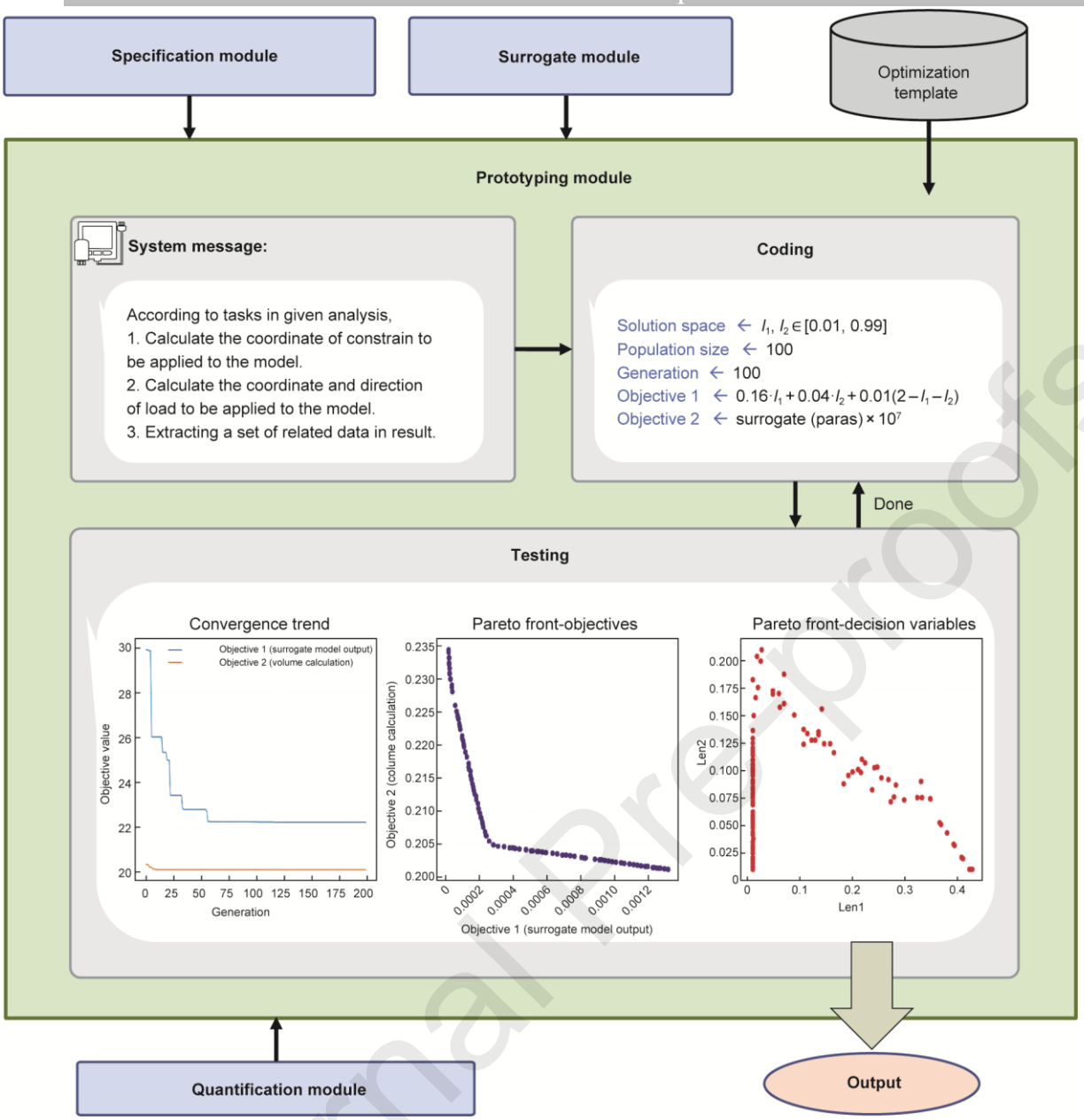


Fig. 7. Workflow of the prototyping module. The verified quantification scripts and surrogate models are encapsulated into a standard Python multi-objective optimization (PyMOO) template (e.g., NSGA-II) for final execution.

Finally, to ensure a robust and standardized optimization, the framework encapsulates the verified quantification scripts and surrogate models into a solver interface. As shown in Fig. 7, the module leverages the Python multi-objective optimization (PyMOO) library. The prototyping module selects the most appropriate algorithm template (e.g., NSGA-II for multi-objective problems) and fills in the blank slots in the template with the specific objective functions generated in the previous steps. Algorithm 3 formalizes the final integration. It uses the surrogate interface (I) and optimization templates (t) as inputs to generate the final solver code. Similar to previous modules, it employs an iterative feedback loop; however, the validation focus shifts to the optimization result (res) and the convergence behavior of the selected algorithm. The Testing block then runs the packaged code to obtain the Pareto front. Similar to the previous modules, if the optimization fails to converge, feedback is provided to dynamically adjust the hyperparameters of the algorithm.

Algorithm 3: prototyping

Input: optimization Task T, surrogate model interface I, hyperparameters HP, code generation prompt $\mathcal{P}_1$, execution prompt $\mathcal{P}_2$, optimization template t

Output: optimization result res

Iteration count initialization: iteration ← 0;
  Decision variable initialization: pass ← false;
  for $i = 1$ to n **do**
    while NOT pass AND iteration < 5 **do**
      Iteration count setting: iteration ← iteration + 1
      Error message initialization: error ← None;
      Code generation: code, $H \leftarrow \text{llm}(\text{Data}, T, t, \mathcal{P}_1, \text{advice})$;
      Code execution: res, error, convergenceCurve ← Execute(code, HP);
      If error = None **then**
        Result validation: $pass, advice \leftarrow \text{llm}(\text{error}, \text{convergenceCurve}, \mathcal{P}_2)$;
        if pass **then**
          Return res;
        end if
      end if
    end while
  end for

4. Experimental results and discussion

As the first framework to achieve fully automated design optimization, we need to validate the following issues: ① Can LLM-IDA accomplish more practical optimization tasks? ② What advantages does LLM-IDA hold over the RAG method, which uses LLMs to solve vertical domain problems? ③ Can LLM-IDA significantly reduce the engineering setup overhead compared to traditional manual workflows? And ④ are the individual components within LLM-IDA truly necessary?

In terms of experimental design, for Issue 1, we utilized the LLM-IDA in Section 4.1, to conduct fully automated structural optimization on a ten-bar beam lightweight problem, a metallic bellows lightweight problem, a lightweight motorcycle framework, and a vehicle side impact problem that has already undergone mathematical modeling to demonstrate the feasibility and transferability of the LLM-IDA in highly automated optimization. For Issue 2, we established a dataset containing diverse industrial design scenarios to evaluate the stability of the LLM-IDA, and compare it with the RAG method integrated with different LLMs in Section 4.2. For Issue 3, we conducted a quantitative efficiency analysis in Section 4.3, specifically comparing the workflow setup overhead (strictly excluding solver execution time) between the LLM-IDA framework and traditional manual workflows to validate the reduction in engineering costs. Finally, we performed ablation experiments to verify the effectiveness of the quantification module and surrogate model within the LLM-IDA in Section 4.4.

4.1. Case studies

In the above case, the loads and constraints were automatically generated by the LLMs based on their internal knowledge. Similarly, load positions can be customized through personalized prompts to achieve specific optimization objectives. A ten-bar truss model is shown in Fig. 8. This structure consists of ten beam elements connected by welding, with the horizontal length of the left half being l, the width of the beams on the left side being w_1, and the beam width of the beams on the right side being w_2. These three variables constitute the optimization variables for the problem. Notably, a special treatment is applied at the welding joints, making it difficult for the LLM to directly derive the structure volume through mathematical modeling. Therefore, incorporated into the CAE-based quantification process. In this section, we adopt the prompting framework used in the three-section beam example. Since the entire workflow has already been clearly demonstrated, only the key results are presented here. Fig. 9 illustrates the results of the multi-objective optimization with personalized prompts, illustrating that the LLM-IDA successfully optimized the ten-bar truss.

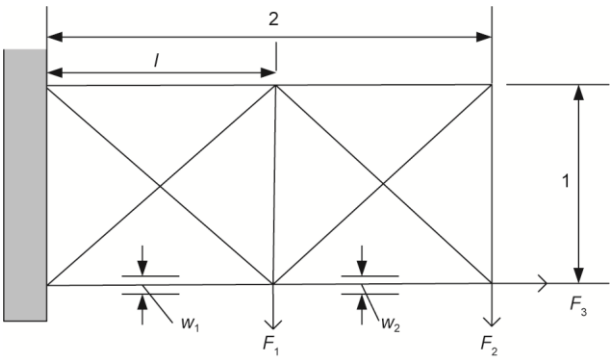


Fig. 8. Geometry and design variables of the ten-bar truss optimization problem.

$$\begin{aligned} \min \quad & \{f_1(\boldsymbol{x}), f_2(\boldsymbol{x})\} = \{Volume(\boldsymbol{x}), MaxDisplacement(\boldsymbol{x})\} \\ \text{s.t.} \quad & \boldsymbol{x} = [l, w_1, w_2]^T \\ & 0.05 \le l \le 0.15 \\ & 0.01 \le w_i \le 0.02, i = 1,2 \end{aligned} \tag{9}$$

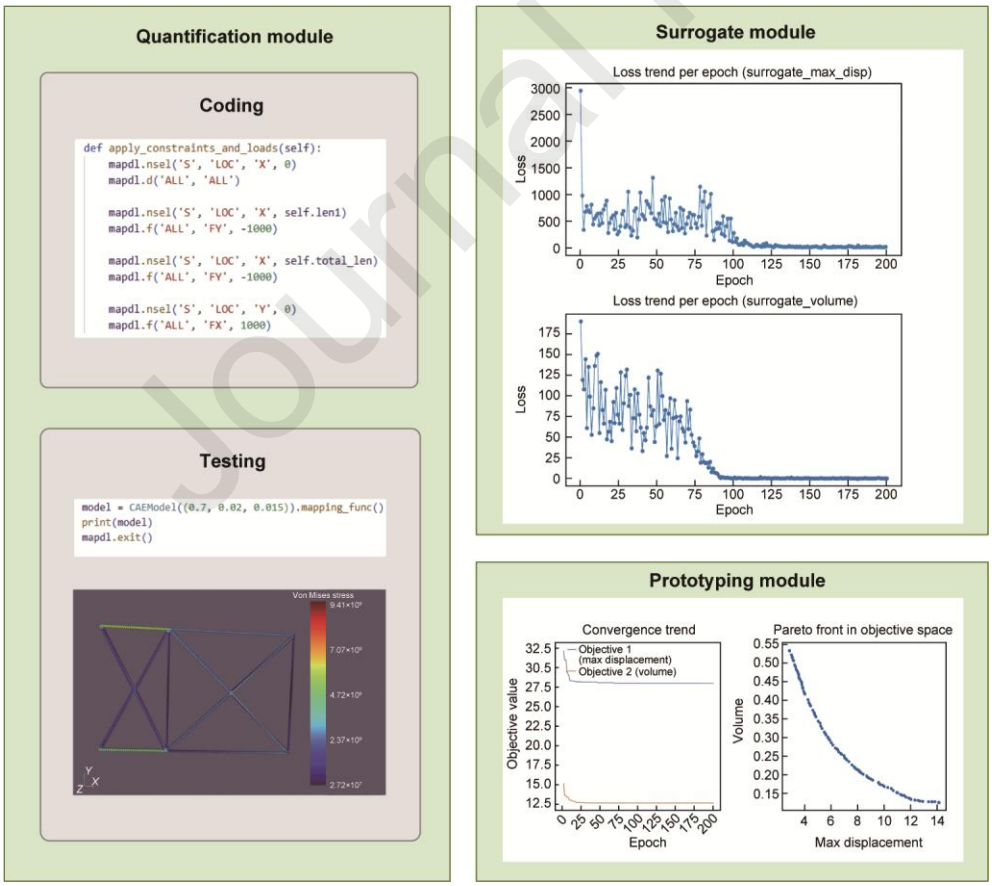


Fig. 9. Optimization workflow and resulting Pareto front for the ten-bar truss using LLM-IDA.

We also applied the framework to a metallic bellows to test its performance. In this scenario, the LLM autonomously managed the parametric modeling of the wall thickness (t_w) and convolution height (h_c) to minimize the volume (in m^3) and maximum stress (in MPa). The optimization problem is mathematically formulated as follows:

$$\begin{aligned} \min \quad & f_1(\boldsymbol{x}), f_2(\boldsymbol{x}) = \text{Volume}(\boldsymbol{x}), \text{MaxStress}(\boldsymbol{x}) \\ \text{s.t.} \quad & \boldsymbol{x} = [t_w, h_c]^T \\ & 0.01 \le t_w \le 0.05 \\ & 0.10 \le h_c \le 0.25 \end{aligned} \tag{10}$$

The comprehensive workflow from automated coding to the derivation of the Pareto front (Fig. 10) validates the robustness of the method for mechanical design.

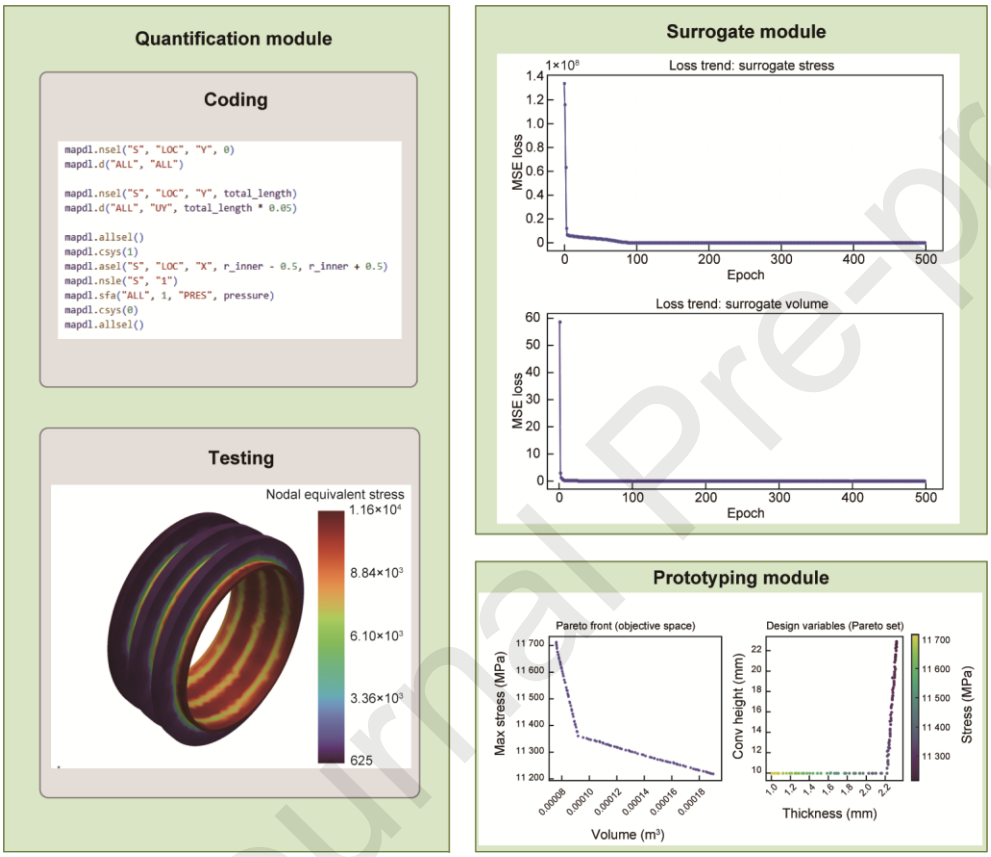


Fig. 10. LLM-IDA lightweight optimization process of the ten-bar truss.

To demonstrate the adaptability of the framework across diverse industrial software platforms, we replaced the prompt template with the API of SolidWorks Simulation and optimized the motorcycle frame for a lightweight design using the mathematical expression in Eq. (11). d_1 is the main pipe diameter, d_2 is branch pipe diameter, and p_t is the position of the tripod.

$$\begin{aligned} \min \quad & f(\boldsymbol{x}) = Volume(\boldsymbol{x}) \\ \text{s.t.} \quad & \boldsymbol{x} = \left[d_1, d_2, p_t\right]^T \\ & MaxStress(\boldsymbol{x}) - 125 \le 0 \\ & 0.02 \le d_1, d_2 \le 0.10, 0.45 \le p_t \le 0.55 \end{aligned} \tag{11}$$

It is important to note that to simulate diverse industrial design scenarios, the input is not a 3D modeling code, but a completed 3D model. This implies that the LLM-IDA cannot read the coordinates of the load and constraint points. Therefore, a manual step is required to select the load points based on LLM-IDA simulation suggestions, as shown in Fig. 11. After completing the simulation, a surrogate model was successfully constructed and optimized. The optimization results for the motorcycle, shown in

Table 1, reveal that the LLM-IDA reduced the motorcycle volume by 39.6% without exceeding the specified stress limits. This experiment demonstrates the potential of LLM-IDA in real-world industrial contexts. If future research enables the identification and selection of the simulation load and constraint points, LLM-IDA can achieve full-process automation in complex scenarios.

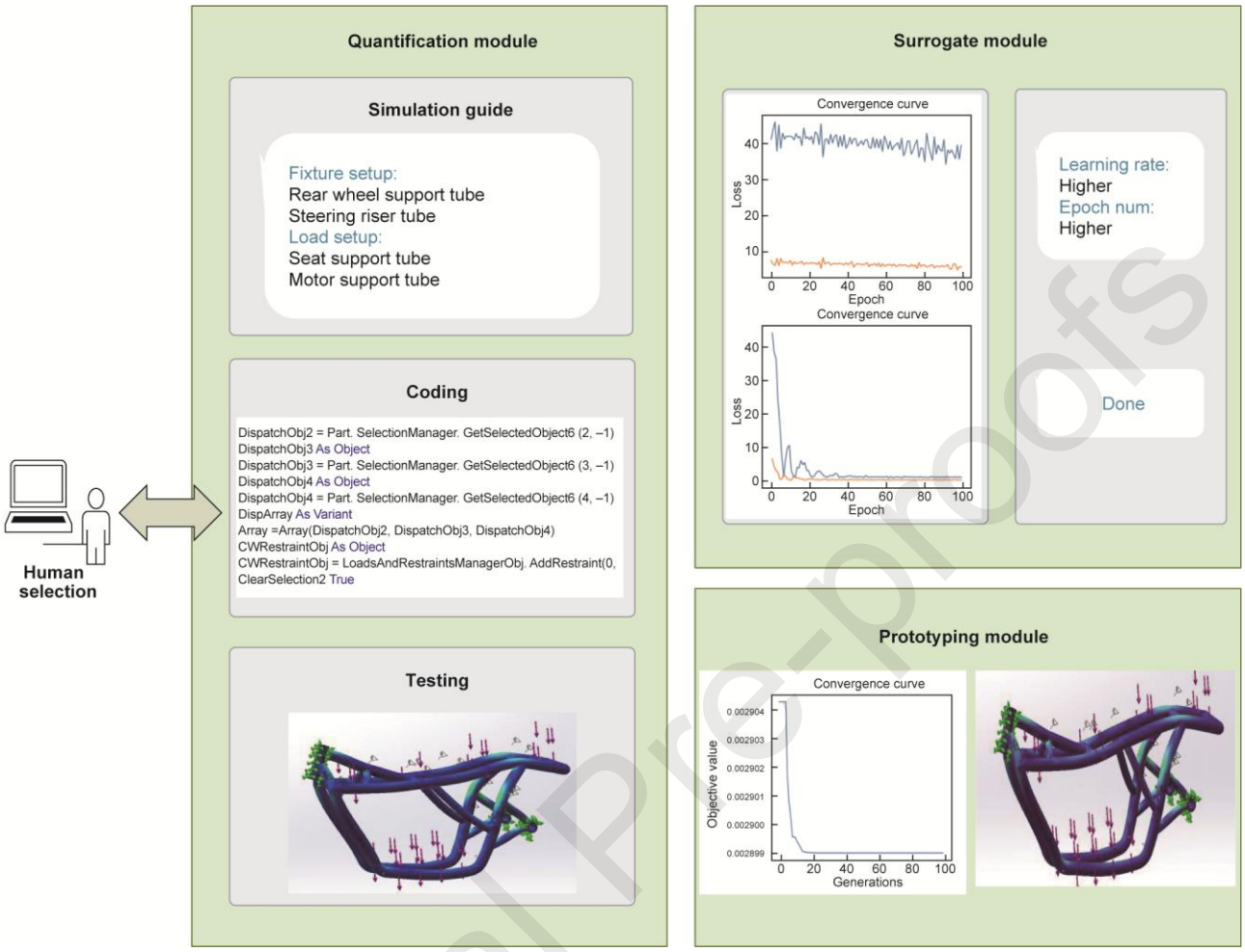


Fig. 11. LLM-IDA lightweight optimization process of the motorcycle frame. Num: number.

Table 1

Optimization results for the motorcycle frame, comparing design parameters, volume (m^3), and maximum stress (MPa) before and after optimization.

Parameter	Analysis result	
	Before optimization	After optimization
Main pipe diameter (m)	0.06	0.03195
Branch pipe diameter (m)	0.04	0.016912
Tripod position (m)	0.055	0.47
Volume (m^3)	0.004797	0.002899 ↓ **39.6%**
Maximum equivalent stress (MPa)	87.9	124.71

Moreover, LLM-IDA is equally capable of solving optimization problems that have already undergone mathematical modeling. As a case study, we refer to the vehicle collision problem using the model depicted in Fig. 12(a). In this problem, the variables to be optimized include the thickness of the inner side of the B-pillar, the B-pillar reinforcement, the inner side of the floor panel, the crossbeam, the door beam, the waistline reinforcement of the door, the roof rail (x_1 to x_7), the material of the inner side of the B-pillar and the inner side of the floor panel (x_8 and x_9), and the obstacle height and impact position (x_{10} and x_{11}). The mathematical expression for this issue is depicted in Eq. (12), with the optimization outcomes compared with distinct algorithms [46] in Table 2, accompanied by a convergence curve illustrated in Fig. 12(b). The results demonstrate that when employing the

LLM-IDA as a black-box optimization tool, its capability to autonomously select algorithms and configure optimization parameters yields outcomes largely consistent with those derived from manual selection and configuration. Therefore, substituting human intervention into the algorithm initialization with LLM is viable.

$$
\begin{aligned}
&\min: f(\boldsymbol{x}) = 1.98 + 4.90x_1 + 6.67x_2 + 6.89x_3 + 4.01x_4 + 1.78x_5 + 2.73x_7 \\
&\text{s.t.:} \\
&\begin{cases}
\boldsymbol{x} = [x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8, x_9, x_{10}, x_{11}]^{\mathrm{T}} \\
0.5 \le x_1, x_2, x_3, x_4, x_5, x_6, x_7 \le 1.5 \\
x_8, x_9 \in \{0.192, 0.345\} \\
-30 \le x_{10}, x_{11} \le 30 \\
g_1(\boldsymbol{x}) = 1.16 - 0.3717x_2x_4 - 0.00931x_2x_{10} - 0.484x_3x_9 + 0.01343x_6x_{10} \le 1 \\
g_2(\boldsymbol{x}) = 33.86 + 2.95x_3 + 0.1792x_{10} - 5.057x_1x_2 - 11.0x_2x_8 - 0.0215x_5x_{10} - 9.98x_7x_8 \\
+22.0x_8x_9 \le 32 \\
g_3(\boldsymbol{x}) = 28.98 + 3.818x_3 - 4.2x_1x_2 + 0.0207x_5x_{10} + 6.63x_6x_9 - 7.7x_7x_8 + 0.32x_9x_{10} \le 32 \\
g_4(\boldsymbol{x}) = 0.261 - 0.0159x_1x_2 - 0.188x_1x_8 - 0.019x_2x_7 + 0.0144x_3x_5 + 0.0008757x_5x_{10} \\
+0.08045x_6x_9 + 0.00139x_8x_{11} + 0.00001575x_{10}x_{11} \le 0.32 \\
g_5(\boldsymbol{x}) = 0.214 + 0.00187x_5 - 0.131x_1x_8 - 0.0704x_1x_9 + 0.03099x_2x_6 - 0.018x_2x_7 \\
+0.0208x_3x_8 + 0.121x_3x_9 - 0.00364x_5x_6 + 0.0007715x_5x_{10} \le 0.32 \\
g_6(\boldsymbol{x}) = 46.36 - 9.9x_2 - 12.9x_1x_8 + 0.1107x_3x_{10} \le 32 \\
g_7(\boldsymbol{x}) = 0.74 - 0.61x_2 - 0.163x_3x_8 + 0.001232x_3x_{10} - 0.166x_7x_9 + 0.227x_2^2 \le 0.32 \\
g_8(\boldsymbol{x}) = 4.72 - 0.5x_4 - 0.19x_2x_3 - 0.0122x_4x_{10} + 0.009325x_6x_{10} + 0.000191x_{11}^2 \le 4 \\
g_9(\boldsymbol{x}) = 10.58 - 0.674x_1x_2 - 1.95x_2x_8 + 0.02054x_3x_9 - 0.0198x_4x_{10} + 0.028x_6x_9 \le 9.9 \\
g_{10}(\boldsymbol{x}) = 16.45 - 0.489x_3x_7 - 0.843x_5x_6 + 0.0432x_9x_{10} - 0.0556x_9x_{11} + 0.000786x_5x_{10} \le 15.7
\end{cases}
\end{aligned}
\tag{12}
$$

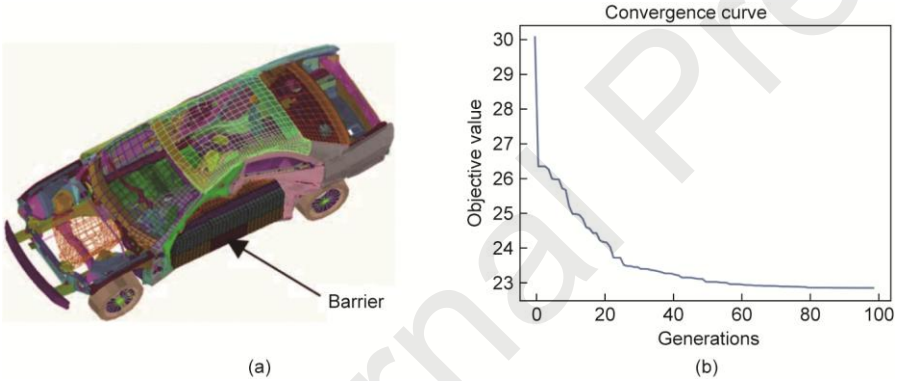


Fig. 12. Vehicle Side Impact Problem. (a) Vehicle side impact model; (b) optimization result of vehicle side impact model.

Table 2
Optimization results for the vehicle side-impact problem, comparing performance metrics (kg) across different algorithms.
Note:

Algorithm	Best	Mean	Worst	Standard deviation
LLM-IDA	**22.84075**	**22.89082**	**23.84210**	**0.218**
PSO	22.85673	23.61240	25.73265	0.138
DE	22.85517	23.12172	23.39516	0.049
FA	22.84318	23.75213	26.19500	0.166
CS	22.94438	24.32517	26.36105	0.252
GWO	22.84647	23.10978	23.49137	0.048
ALO	22.84622	23.39635	23.89138	0.100

MFO	22.84297	23.12039	23.48124	0.051
WOA	23.19722	24.33408	26.34446	0.208
RAO-3	23.23209	24.37533	28.42530	0.209
MGO	22.84324	23.14917	23.30040	0.056

FA: Firefly Algorithm; CS: Cuckoo Search; GWO: Grey Wolf Optimizer; ALO: Ant Lion Optimizer; MFO: Moth Flame Optimization; WOA: Whale Optimization Algorithm; RAO-3: Rao Algorithm 3; MGO: Mountain Gazelle Optimizer.

4.2. Pass@10 evaluation

The automated simulation and optimization tasks we undertake are fundamentally code-generation tasks. The task analysis and decomposition performed prior to code generation define its ultimate objective. To validate the feasibility and cross-domain applicability of this framework, we curated a diverse dataset comprising 15 industrial case studies, including ten truss structures with varying angles and cross sections, three pressure vessels, a robotic arm manipulator, and a microelectromechanical system (MEMS) switch. All cases were subjected to automated lightweight design optimization. Since the framework includes three code generation modules, we defined a total of four evaluation tasks, with the success criteria for each task outlined as follows: ① objective-quantification test, where success requires the successful execution of the code and the correct configuration of both pre-processing and post-processing settings; ② surrogate model construction test; ③ optimization test, where success is defined by the correct code execution and completion of the optimization loop; and ④ system integration test, where all three preceding tasks must be successfully executed in a single run, culminating in the objectives converging and achieving a Pareto front.

It is important to clarify that the optimization algorithms (NSGA-II and PSO) utilized in Tasks 3 and 4 are employed as standard off-the-shelf components invoked by the LLM-generated scripts. Consequently, our evaluation focuses on "automation fidelity"—the framework's ability to correctly configure and execute these algorithms—rather than intrinsic algorithmic metrics such as convergence time or hyperparameter sensitivity. Therefore, for these four tasks, we utilized test cases to evaluate the precise functional correctness of the generated code and measured the performance using pass@k [45], ensuring a certain level of real-world applicability.

$$\text{pass@10} = \mathbb{E}_{\text{problems}}\left[1 - \frac{\binom{n-c}{10}}{\binom{n}{10}}\right], \quad n = 50 \tag{13}$$

where n represents the total number of generations, and c is the number of samples that passed all test cases, the metric is evaluated with a fixed sample size of 10. $1 - \frac{\binom{n-c}{10}}{\binom{n}{10}}$ provides an estimated pass@10 for an individual problem and $\mathbb{E}$ represents the expected pass@10 across all problems.

Fig. 13 illustrates a comparative analysis between the LLM-IDA method (utilizing GPT-4-Turbo) and various current code-generation models augmented with the RAG method to validate the performance of the framework. For the LLM-IDA framework, we applied differentiated hyperparameter settings: The temperature was set to 0.7 for the Task Analysis LLM to encourage reasoning diversity, and 0.3 for the Code Generation LLM to ensure syntactic precision. For the baseline RAG implementation, we utilized a vector similarity retrieval strategy employing Doubao-Embedding for semantic embedding and Gte-Rerank-V2 for result ranking, with the retrieval depth fixed at Top-$k = 3$. Under these controlled conditions, the LLM-IDA method achieved a substantial improvement in pass@10 across the four evaluation tasks compared with the RAG method.

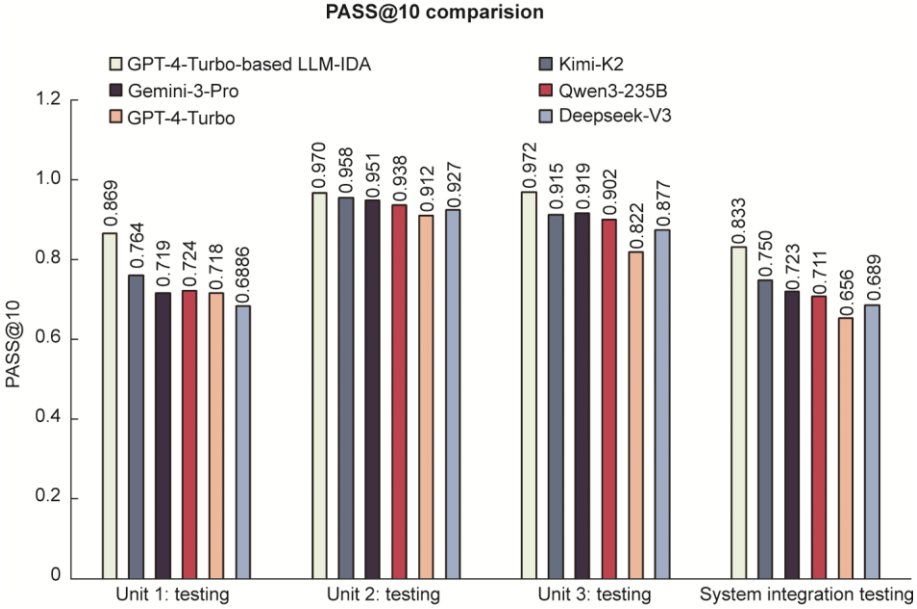


Fig. 13. Pass@10 of four tasks.

Notably, the performance enhancement was most significant in Unit Tests 1 and 3 (i.e., the quantification and prototyping modules). This is primarily because PyAnsys and PyMOO are domain-specific libraries with relatively sparse training corpora for general LLMs, making them prone to hallucinations. In this context, the "Task Analysis" and "Template Filling" mechanisms in LLM-IDA effectively perform dimensionality reduction for the code generation task. By explicitly defining which code blocks are mutable based on problem analysis and which must strictly adhere to immutable templates, our framework eliminates most of the API hallucinations. For instance, whereas baseline models frequently generate non-existent parameters, such as Gemini-3-Pro passing an allow_non_interactive argument to the launch_mapdl method, or Qwen3-235B inserting a title parameter into the plot_principal_nodal_stress function—LLM-IDA structurally prevents such errors. These empirical results strongly corroborate the theoretical propositions in Eq. (4).

4.3. Efficiency analysis: Automated vs. manual workflows

With the aim of objectively quantifying the industrial value of the proposed framework, we conducted a comparative study focusing on "process efficiency" rather than "solver efficiency." We introduced a specific metric, workflow overhead (T_{overhead}), defined as the total time required to formulate, implement, and debug the optimization problem, excluding the computational solution time (T_{solver}).

$$T_{\text{total}} = T_{\text{overhead}} + T_{\text{solver}} \tag{14}$$

Because LLM-IDA invokes the same standard off-the-shelf solvers (e.g., ANSYS MAPDL) as a manual workflow, T_{solver} remains constant for identical physics. Therefore, the efficiency gain is determined solely by the reduction in T_{overhead}.

We compared the T_{overhead} of the LLM-IDA against an intermediate-level simulation engineer (with over three years of hands-on experience in Ansys and proficiency in the PyTorch and PyMOO frameworks) for two representative case studies.

Table 3
Comparison of workflow overhead. T_{manual} (manual overhead) represents estimated time for requirement analysis, API documentation lookup, script coding, and iterative debugging. T_{auto} (LLM-IDA overhead) represents Mean time $\pm$ standard deviation over 5 independent runs, including LLM inference latency, task specification, and dynamic self-correction loops.

Case study	Manual overhead (T_{manual}, s)	LLM-IDA overhead (T_{auto}, s)	Efficiency gain
Three-section Cantilever beam	1024	181 ± 39	5.66 ×
Ten-bar truss	1740	263 ± 45	6.61 ×
Pressure vessel	1560	303 ± 40	5.15 ×
Average	1441	249 ± 41	5.79 ×

Table 3 provides the empirical evidence that the LLM-IDA framework effectively eliminates human bottlenecks. LLM-IDA completes the entire code generation and validation process in approximately 5 min (approximately 82.7% reduction in overhead). The variance in automation time is primarily attributed to fluctuations in the API latency and the number of self-correction iterations required.

4.4. Ablation study

To evaluate the individual contributions of the Task Analysis, Code Verification, and Built-in Template modules, we conducted an ablation study using four representative base LLMs: GPT-4-Turbo, DeepSeek-Coder-V2, Gemini-1.5-pro(175B), and Codestral. These models were strategically chosen to span a wide spectrum of parameter scales and long-context reasoning capabilities, enabling us to examine how each module enhances the models with varying intrinsic strengths using Pass@10 as the metric. We established three experimental groups: ① With and without Built-in Templates (compared against a strong RAG baseline); ② With and without Task Analysis; and ③ With and without Code Testing. GPT-4o was uniformly employed as the visual inspector in the verification step to ensure multimodal support across all baselines.

(1) Effect of Built-in Templates: As shown in the first section of Table 4, LLM-IDA consistently outperforms the RAG method. The data reveal a positive correlation between model reasoning capability and performance gain: models with stronger long-context reasoning (e.g., GPT-4-Turbo +14.12%, DeepSeek-Coder-V2 +12.58%) benefit significantly more than smaller models (e.g., Codestral +3.69%). This indicates that the structural templates effectively leverage the latent simulation knowledge of advanced LLMs.

(2) Effect of Task Analysis: The second section of Table 4 compares frameworks with and without the Task Analysis module. Interestingly, smaller models exhibited larger relative gains. This corroborates the theory expressed in Eq. (2), suggesting that Task Analysis functions as a CoT mechanism. This guides parameter-constrained models to decompose obscure PyAnsys coding tasks into fundamental mechanical and geometric problems, thereby bridging the reasoning gap in which the training corpora are sparse.

(3) Effect of Code Testing: The quantitative improvement is presented in the final section of Table 4, which provides a self-correction feedback loop, allowing the framework to rectify errors arising from the initial analysis or generation, thus ensuring robust execution.

Table 4
Ablation study results showing pass@10 performance with and without key modules across different LLM models.

Method	GPT-4-Turbo	DeepSeek-Coder-V2 (236B)	Codestral (22B)	Gemini-1.5-pro (175B)
With built-in templates	0.889	0.85	0.506	0.558
Without built-in templates	0.779	0.755	0.488	0.434
Improvement rate	+14.121%	+12.583%	+3.689%	+28.571%
With task analysis	0.889	0.85	0.506	0.558
Without task analysis	0.664	0.651	0.322	0.441
Improvement rate	+33.886%	+30.568%	+57.143%	+26.531%
With code testing	0.889	0.85	0.506	0.558
Without code testing	0.816	0.803	0.491	0.451
Improvement rate	+8.946%	+5.853%	+3.055%	+23.725%

The computational efficiency of the LLM-IDA automatic surrogate model construction was validated through a quantitative assessment using the three-section cantilever beam and ten-bar truss problems. Regarding the hardware setup, all FEA simulations were executed on a workstation equipped with an Intel Core i9-13900K CPU. LLM reasoning was performed using the GPT-4-Turbo API, involving an average of 5–8 API calls per task.

The quality of the Pareto fronts was evaluated using the Hypervolume (HV) metric, which measures the volume of the objective space dominated by the solution set $\mathcal{A}$ with respect to a reference point r. Mathematically, it is defined as the Lebesgue measure Λ of the union of hypercubes:

$$\mathrm{HV}(\mathcal{A}, r) = \Lambda\left(\bigcup_{y \in \mathcal{A}} \{z \in \mathbb{R}^d \mid y < z < r\}\right) \tag{15}$$

where d is the number of objectives, y is a Pareto solution in the objective space, and z is a point in the d-dimensional objective space. Furthermore, to quantify the accuracy loss introduced by surrogate modeling, we defined the HV deviation ($\mathrm{Dev_{HV}}$) metric as follows:

$$\mathrm{Dev_{HV}} = \frac{\mathrm{HV_{baseline}} - \mathrm{HV_{surrogate}}}{\mathrm{HV_{baseline}}} \times 100\% \tag{16}$$

where $\mathrm{HV_{baseline}}$ represents the HV of the converged non-surrogate Pareto front.

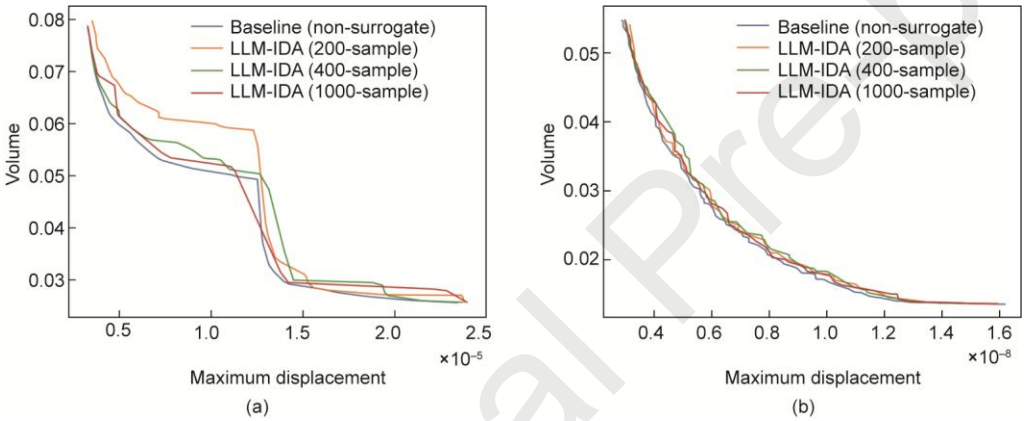


Fig. 14. Pareto fronts obtained with different sampling budgets for the surrogate models. (a) Three-section cantilever beam; (b) ten-bar truss. The results show that the Pareto fronts generated by LLM-IDA with 200, 400, and 1000 samples closely match those of the baseline (non-surrogate) approach.

For the baseline (non-surrogate) approach, we set the maximum population size to 100. The optimization processes converged at the 52nd generation for the cantilever beam and the 61st generation for the ten-bar truss, resulting in total actual simulation counts of 5200 and 6100, respectively. In contrast, LLM-IDA autonomously constructed surrogate models using fixed sampling budgets of 200, 400, and 1000 simulations.

Fig. 14 shows that the Pareto fronts generated by LLM-IDA closely align with the baseline. Quantitatively (see Table 5), even when compared to the actual converged baseline, LLM-IDA achieved a reduction in simulation time of up to 96.71% (in the Ten-bar Truss 200-sample case) while maintaining a $\mathrm{Dev_{HV}}$ of less than 2.76% across all tested scenarios. This confirms that the framework effectively balances the computational cost and optimization accuracy.

Table 5
Quantitative comparison of FEA simulation cost and optimization accuracy (HV) between baseline and LLM-IDA methods.

Optimization problem	Method	Sampling size (FEA calls)	Total FEA simulation time (s)	Simulation Time Savings	HV	$\mathrm{Dev_{HV}}$
Cantilever beam	Baseline (non-surrogate)	5200	16536	—	0.832	—

(converged at Gen 52)						
	LLM-IDA (1000-sample)	1000	3180	80.77%	0.823	1.08%
	LLM-IDA (400-sample)	400	1272	92.31%	0.822	1.20%
	LLM-IDA (200-sample)	200	636	96.15%	0.809	2.76%
Ten-bar truss	Baseline (non-surrogate)	6100	11102	—	0.652	—
(converged at Gen 61)						
	LLM-IDA (1000-sample)	1000	1820	83.61%	0.641	1.69%
	LLM-IDA (400-sample)	400	728	93.44%	0.635	2.61%
	LLM-IDA (200-sample)	200	365	96.71%	0.638	2.15%

5. Conclusion and outlook

5.1. Conclusion

To assist human engineers in industrial design automation, this paper presents the LLM-IDA framework, which encompasses conceptual design, knowledge-based design, and digital prototyping. This framework uses a multitiered AI agent group for four modular processes: ① specification, ② quantification, ③ surrogate, and ④ prototype module. With specification and code generation capabilities of LLMs, the LLM-IDA effectively delivers industrial design automation. A comparison with state-of-the-art RAG methods using the pass@10 metric demonstrates that the LLM-IDA framework provides greater generality and reliability with improved time efficiency and optimality over existing domain-specific methods.

In summary, LLM-IDA presents the following advantages.

(1) Automation: LLM-IDA offers a framework to achieve fully automated engineering designs, where error detection and feedback throughout the process are handled entirely by the LLM without human intervention;

(2) Surrogate construction: LLM-IDA features automatic surrogate model construction, significantly speeding up the optimization process;

(3) Cross-domain applicability: Although validated primarily for mechanical problems, the framework's modular architecture is inherently domain agnostic, where the LLM-IDA can be extended to electronic design automation, energy systems, and materials engineering, for example, by substituting the underlying physics kernels (e.g., ANSYS Maxwell or Fluent).

5.2. Limitation and future work

Currently, the LLM-IDA relies on general-purpose LLMs, which are not specifically oriented towards the physical world. Consequently, there exist limitations primarily in: ① Limited spatial perception: The difficulty LLMs face in perceiving complex CAD geometries hinders the formulation of geometry-dependent optimization objectives, and ② long-context instability: The inherent stochasticity of LLMs in long-chain reasoning poses challenges to system reliability.

In future work, we aim to address these limitations by fine-tuning open-source models on CAD/CAE datasets to enhance reasoning accuracy and integrating multimodal capabilities to enable an intuitive understanding of CAD design inputs. More fundamentally, we aim to develop "AI for Engineering" world models.

Acknowledgement

This work was support in part by the National Natural Science Foundation of China under grants W2431044, 92270105, and 52405253 and in part by the Ministry of Science and Technology of China under grant H20240917.

7. Reference

[1] Ang JH, Goh C, Li Y. Smart design for ships in a smart product through-life and industry 4.0 environment. In: Proceedings of the 2016 IEEE Congress on Evolutionary Computation (CEC); 2016 Jul 24–29; Vancouver, BC, Canada. New York City: IEEE; 2016. p. 5301–8.

[2] Chen Y, Li Y. Computational intelligence assisted design: in industrial revolution 4.0. Boca Raton: CRC Press; 2018.

[3] Brand J, Israeli A, Ngwe D. Using LLMs for market research. In: Harvard business school marketing unit working paper. Boston: Harvard Business School; 2023. p. 23-062.
[4] Arora C, Grundy J, Abdelrazek M. Advancing requirements engineering through generative AI: assessing the role of LLMs. In: Generative AI for effective software development. Cham: Springer Nature Switzerland; 2024. p. 129–48.
[5] Xu S, Wei Y, Zheng P, Zhang J, Yu C. LLM enabled generative collaborative design in a mixed reality environment. J Manuf Syst 2024;74:703–15.
[6] Huang L, Yu W, Ma W, Zhong W, Feng Z, Wang H, et al. A survey on hallucination in large language models: principles, taxonomy, challenges, and open questions. ACM Trans Inf Syst 2025;43(2):1–55.
[7] Jiang G, Ma Z, Zhang L, Chen J. EPlus-LLM: a large language model-based computing platform for automated building energy modeling. Appl Energy 2024;367:123431.
[8] Wei J, Wang X, Schuurmans D, Bosma M, Ichter B, Xia F, et al. Chain-of-thought prompting elicits reasoning in large language models. Adv Neural Inf Process Syst 2022;35:24824–37 .
[9] Lewis P, Perez E, Piktus A, Petroni F, Karpukhin V, Goyal N, et al. Retrieval-augmented generation for knowledge-intensive NLP tasks. Adv Neural Inf Process Syst 2020;33:9459–74 .
[10] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need. In: Advances in neural information processing systems. 2017. p. 30.
[11] Devlin J, Chang MW, Lee K, Toutanova K. Bert: pre-training of deep bidirectional transformers for language understanding. Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers); 2019 Jun 2–7; Minneapolis, MN, USA. Stroudsburg: Association for Computational Linguistics; 2019. p. 4171–86.
[12] Brown T, Mann B, Ryder N, Subbiah M, Kaplan J, Dhariwal P, et al. Language models are few-shot learners. Adv Neural Inf Process Syst 2020;33:18771901 .
[13] Radford A, Narasimhan K, Salimans T, Salimans T, Sutskever I. Improving language understanding by generative pre-training. In: Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2018; 2018 Jun 1–6; New Orleans, LA, USA. Kerrville: Association for Computational Linguistics; 2018.
[14] Radford A, Wu J, Child R, Luan D, Amodei D, Sutskever I. Language models are unsupervised multitask learners. Report. San Francisco: OpenAI Blog; 2019.
[15] Ramesh A, Pavlov M, Goh G, Gray S, Voss C, Radford A, et al. Zero-shot text-to-image generation. PMLR 2021;139:8821–31.
[16] Rombach R, Blattmann A, Lorenz D, Esser P, Ommer B. High-resolution image synthesis with latent diffusion models. In: Proceedings of the IEEE/CVF Conference on Computer Vision And Pattern Recognition (CPVR 2022); 2022 Jun 18–24; New Orleans, LA, USA. New York City: IEEE; 2022. p. 10684–95.
[17] Hu EJ, Shen Y, Wallis P, Allen-Zhu Z, Li Y, Wang S, et al. LoRA: low-rank adaptation of large language models. 2022. arXiv:2106.09685.
[18] Houlsby N, Giurgiu A, Jastrzebski S, Morrone B, De Laroussilhe D, Gesmundo A, et al. Parameter-efficient transfer learning for NLP. PMLR 2019;97:2790–9.
[19] Li X L, Liang P. Prefix-tuning: optimizing continuous prompts for generation. In: Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers). 2021. p. 4582–97.
[20] Wang L, Ma C, Feng X, Zhang Z, Yang H, Zhang J, et al. A survey on large language model based autonomous agents. Front Comput Sci 2024;18(6):186345.
[21] Ning KP, Yao JY, Liu YY, Ning MN, Yuan L. GPT as a monte Carlo language tree: a probabilistic perspective. 2025. arXiv:2501.07641.
[22] Wu H, He Z, Zhang X, Yao X, Zheng S, Zheng H, et al. Chateda: a large language model powered autonomous agent for eda. IEEE Trans Comput Aided Des Integrated Circ Syst 2024;43(10):3184–97.
[23] Wu S, Khasahmadi A, Katz M, Jayaraman PK,·Pu Y,·Willis K, et al. CAD–LLM: large language model for cad generation. In: Proceedings of the 37th Neural Information Processing Systems Conference (NeurIPS 2023); 2023 Dec 10–16; New Orleans, LA, USA. Red Hook: Curran Associates Inc.; 2023.
[24] Lin SW, Ying KC, Chen SC, Lee ZJ. Particle swarm optimization for parameter determination and feature selection of support vector machines. Expert Syst Appl 2008;35(4):1817–24.
[25] Holland JH. Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. Cambridge: MIT Press; 1992.
[26] Kirkpatrick S, Gelatt Jr CD, Vecchi MP. Optimization by simulated annealing. Science 1983;220(4598):671–80.
[27] Storn R, Price K. Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. J Glob Optim 1997;11(4):341–59.
[28] Pereira JLJ, Oliver GA, Francisco MB, Cunha SS Jr, Gomes GF. A review of multi-objective optimization: methods and algorithms in mechanical engineering problems. Arch Comput Methods Eng 2022;29(4):2285–308.
[29] Rao RV, Savsani VJ, Vakharia DP. Teaching–learning-based optimization: a novel method for constrained mechanical design optimization problems. Comput Aided Des 2011;43(3):303–15.
[30] Yildiz AR, Abderazek H, Mirjalili S. A comparative study of recent non-traditional methods for mechanical design optimization. Arch Comput Methods Eng 2020;27(4):1031–48.
[31] Yu S, Wang Z. A general decoupling approach for time-and space-variant system reliability-based design optimization. Comput Methods Appl Mech Eng 2019;357:112608.
[32] Yu S, Wu X, Zhao D, Li Y. A two-level surrogate framework for demand-objective time-variant reliability-based design optimization. Reliab Eng Syst Saf 2024;244:109924.
[33] Bao Z, Watanabe T. A new approach for circuit design optimization using genetic algorithm. In: Proceedings of the 2008 27th Annual IEEE International Soc Design Conference; 2008 Nov 24–25; Busan, South Korea. New York City: IEEE; 2008, 1: I-383-I-386.
[34] Lberni A, Marktani MA, Ahaitouf A, Ahaitouf A. Analog circuit sizing based on evolutionary algorithms and deep learning. Expert Syst Appl 2024;237:121480.
[35] Li J, Zeng Y, Zhi H, Yang J, Shan W, Li Y, et al. Knowledge transfer framework for PVT robustness in analog integrated circuits. IEEE Trans Circuits Syst I Regul Pap 2023;71(5):2017–30.
[36] Li J, He S, Li A, Yu S, Li Y. Multitask evolution with problem reformulation for global exploration in analog circuit design. Adv Eng Inform 2026;70:104195.
[37] Yang XS. Engineering optimization and industrial applications. In: Koziel S, Leifsson L, editors. Surrogate-based modeling and optimization. New York City: Springer New York; 2013. p. 393–412.
[38] Yang XS, Koziel S. Computational optimization, modelling and simulation–a paradigm shift. Procedia Comput Sci 2010;1(1):1297–300.
[39] Koziel S, Bandler JW, Madsen K. Quality assessment of coarse models and surrogates for space mapping optimization. Optim Eng 2008;9(4):375–91.
[40] Leifsson L, Koziel S. Multi-fidelity design optimization of transonic airfoils using physics-based surrogate modeling and shape-preserving response prediction. J Comput Sci 2010;1(2):98–106.
[41] Koziel S, Koziel S, Yang XS, Zhang QJ. Simulation-driven design optimization and modeling for microwave engineering. London: Imperial College Press; 2013.
[42] Horton JJ. Large language models as simulated economic agents: what can we learn from homo silicus? Report. Cambridge: National Bureau of Economic Research; 2023.

[43] Chen L, Tsang Y, Jing Q, Sun L. A LLM-augmented morphological analysis approach for conceptual design. In: Gray C, Ciliotta Chehade E, Hekkert P, Forlano L, Ciuccarelli P, Lloyd P, editors. Proceedings of the 2024 Design Research Society (DRS) Conference; 2024 Jun 23–28; Boston, MA, USA. London: Design Research Society; 2024.
[44] Qin S, Guan H, Liao W, Gu Y, Zheng Z, Xue H, et al. Intelligent design and optimization system for shear wall structures based on large language models and generative artificial intelligence. J Build Eng 2024;95:109996.
[45] Chen M, Tworek J, Jun H, Yuan Q, de Oliveira Pinto HP, Kaplan J, et al. Evaluating large language models trained on code. 2021. arXiv: 2107.03374.
[46] Sadeeq HT, Abdulazeez AM. Car side impact design optimization problem using giant trevally optimizer. Structures 2023;55:39–45.
[47] Liao J, Xu J, He S, Chen Z, Yu S, Li Y. AutoForma: a large language model-based multi-agent for computer-automated design. In: Proceedings of the 2024 IEEE International Conference on Systems, Man, and Cybernetics; 2024 Oct 7–10; Kuching, Sarawak, Malaysia. New York City: IEEE; 2024. p. 1284–9.

Declaration of Interest Statement

☐ The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

☐ The author is an Editorial Board Member/Editor-in-Chief/Associate Editor/Guest Editor for this journal and was not involved in the editorial review or the decision to publish this article.

☒ The authors declare the following financial interests/personal relationships which may be considered as potential competing interests:

Yun Li and Shui Yu report financial support provided by the National Natural Science Foundation of China and by the Ministry of Science and Technology of China. Other authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.