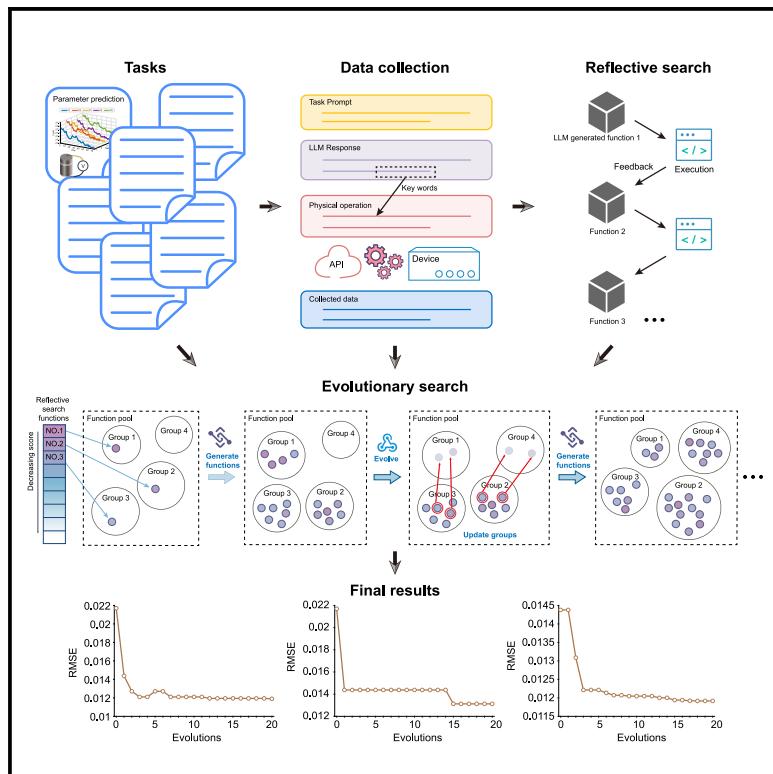


# Patterns

## RePower: An LLM-driven autonomous platform for power system data-guided research

### Graphical abstract



### Highlights

- An automated platform integrates data acquisition, algorithm design, and evolution
- Results improve continuously using a two-stage reflection-evolution search framework
- The framework is validated with six data-driven tasks in the power system
- Our evolutionary strategy outperforms other frameworks in comparative tests

### Authors

Yu-Xiao Liu, Mengshuo Jia,  
Yong-Xin Zhang, Jianxiao Wang,  
Guannan He, Shao-Long Zhong,  
Zhi-Min Dang

### Correspondence

jia@eeh.ee.ethz.ch (M.J.),  
dangzm@tsinghua.edu.cn (Z.-M.D.)

### In brief

The authors present an automatic scientific research platform called RePower, which utilizes a two-stage reflection-evolution strategy for long-context thinking and reasoning to complete data-driven tasks of power systems. This framework can better assist large language models (LLMs) in accomplishing complex research tasks, including physical and virtual ones, further promoting the application of generative artificial intelligence in the field of scientific research.



## Article

# RePower: An LLM-driven autonomous platform for power system data-guided research

Yu-Xiao Liu,<sup>1</sup> Mengshuo Jia,<sup>2,\*</sup> Yong-Xin Zhang,<sup>1</sup> Jianxiao Wang,<sup>3</sup> Guannan He,<sup>4</sup> Shao-Long Zhong,<sup>1</sup> and Zhi-Min Dang<sup>1,5,\*</sup>

<sup>1</sup>State Key Laboratory of Power System Operation and Control, Department of Electrical Engineering, Tsinghua University, Beijing 100084, P.R. China

<sup>2</sup>Power Systems Laboratory, Department of Information Technology and Electrical Engineering, ETH Zürich, 8092 Zürich, Switzerland

<sup>3</sup>National Engineering Laboratory for Big Data Analysis and Applications, Peking University, Beijing 100871, P.R. China

<sup>4</sup>Department of Industrial Engineering and Management, College of Engineering, Peking University, Beijing 100871, P.R. China

<sup>5</sup>Lead contact

\*Correspondence: [jia@eeh.ee.ethz.ch](mailto:jia@eeh.ee.ethz.ch) (M.J.), [dangzm@tsinghua.edu.cn](mailto:dangzm@tsinghua.edu.cn) (Z.-M.D.)

<https://doi.org/10.1016/j.patter.2025.101211>

**THE BIGGER PICTURE** Artificial intelligence (AI) has transformed numerous industries in the 21<sup>st</sup> century, significantly advancing the acquisition, processing, and development of algorithms in scientific research. The rapid growth of the AI industry has led to the emergence of generative artificial intelligence, such as large language models (LLMs) like GPT, which have demonstrated exceptional capabilities in handling language tasks across various fields. Due to their advanced understanding and reasoning abilities, LLMs are increasingly being applied in scientific research and have the potential to drive new discoveries and enhance research efficiency. In existing studies, scientists have already utilized LLMs to automate research in multiple domains, such as chemistry, mathematics, and materials science. If these frameworks can be successfully transferred to the field of power systems or expanded further in other areas, then the full potential of generative AI could be harnessed to support researchers in conducting efficient scientific work.

## SUMMARY

Large language models (LLMs) have shown strong capabilities across disciplines such as chemistry, mathematics, and medicine, yet their application in power system research remains limited, and most studies still focus on supporting specific tasks under human supervision. Here, we introduce Revive Power Systems (RePower), an autonomous LLM-driven research platform that uses a reflection-evolution strategy to independently conduct complex research in power systems. RePower assists researchers by controlling devices, acquiring data, designing methods, and evolving algorithms to address problems that are difficult to solve but easy to evaluate. Validated on three critical data-driven tasks in power systems—parameter prediction, power optimization, and state estimation—RePower outperformed traditional methods. Consistent performance improvements were observed across multiple tasks, with an average error reduction of 29.07%. For example, in the power optimization task, the error decreased from 0.00137 to 0.000825, a reduction of 39.78%. This framework facilitates autonomous discoveries, promoting innovation in power systems research.

## INTRODUCTION

Power systems are among the most critical and complex infrastructures in modern society, consisting of interconnected components that manage the generation, transmission, distribution, monitoring, and regulation of electrical energy.<sup>1</sup> In power systems research, tasks are often structured through modular algorithms or solution frameworks. Exploring the optimal modular algorithm is not a conventional optimization problem, prompting

researchers to turn to artificial intelligence (AI) as a promising approach to tackle this challenge.<sup>2–4</sup>

With the emergence and development of generative AI and large language models (LLMs),<sup>5–7</sup> scientists have begun to build intelligent agents or automated research assistants to assist people in completing complex tasks.<sup>8–11</sup> In the field of chemical research, scholars have proposed and developed intelligent agents such as Coscientist<sup>12</sup> and ChemCrow.<sup>13</sup> These types of agents are capable of performing chemical synthesis planning



and corresponding physical operations, effectively promoting the development and advancement of autonomous scientific research using LLM. In the field of mathematical computation, scholars have proposed Funsearch,<sup>14</sup> an intelligent agent based on feedback-driven autonomous search heuristics, which is capable of finding superior solutions to combinatorial mathematics problems compared to existing ones. However, these agents cannot be directly applied to power systems research. Agents in chemistry are designed for short-text, short-memory-chain reasoning processes within their domain. However, the tasks of power systems often require complex reasoning processes and long chains of thinking and memory. If the design of the prompt and the reasoning process of the LLM is not properly considered, then there is a high likelihood of errors occurring, which could lead to task interruption. While mathematical computation agents can evolve algorithms, their capabilities are confined to combinatorics and do not extend to general problems such as prediction in power systems.

Despite the progress in other fields, the use of LLMs as automated research assistants in power systems remains underexplored. Some scholars have designed agent frameworks for simple tasks in power systems, such as document retrieval or knowledge queries.<sup>15,16</sup> Some of these frameworks include direct feedback mechanisms, enabling multi-step reasoning.<sup>15</sup> However, compared to the field of chemistry, these agents do not integrate all functionalities into a single framework; instead, they design specific frameworks to achieve a particular function, such as document retrieval. Additionally, most of these tasks involve simple question-and-answer interactions with short contexts, and there is no design for feedback on long memory chains. Other scholars have incorporated LLMs as a component in their algorithmic constructions.<sup>17,18</sup> Arnautov and Akimov only used LLM for the parameter design of a three-node power system, including direct feedback on whether the parameters are correct,<sup>17</sup> while Yan and Xu employed a feedback mechanism that evaluates performance based on output results to optimize system performance in real time.<sup>18</sup> However, the generative capabilities of LLMs are not fully utilized; instead, they are used as traditional tools. Furthermore, none of the existing studies involve the control of physical devices,<sup>19,20</sup> which represents a significant gap compared to advancements in the field of chemistry. Consequently, the full potential of LLMs as autonomous research designers in this field has yet to be realized.

Designing an LLM-based automated research assistant for power systems poses distinct challenges due to the complex and often prolonged reasoning required in these tasks.<sup>19,20</sup> Existing LLM research assistants, such as FunSearch in mathematics,<sup>14</sup> are capable of solving predefined tasks with clear structures but struggle with tasks demanding extensive reasoning. Although power systems research tasks can sometimes be reframed as data-driven, end-to-end processes requiring less reasoning,<sup>21–23</sup> they still involve intricate designs. These include challenges such as feature engineering, algorithm development, technique modification, and hyperparameter tuning. Additionally, power systems research often requires interaction with physical equipment, a capability beyond the current scope of LLM-based assistants like FunSearch.

To address the challenges in power systems research and unlock the full potential of LLMs, we introduce RePower

(Revive Power Systems), an auto-evolving research platform designed to autonomously create and refine solution algorithms. RePower operates in two distinct stages. Stage 1 employs self-reflection mechanisms<sup>24</sup> to sequentially explore and design various methods, while stage 2 refines these solutions through function pools and evolutionary algorithms, enabling substantial improvements through both broad (horizontal, prioritizing trying more different methods) and in-depth (vertical, prioritizing improving and adjusting existing methods) explorations. Unlike existing tools like FunSearch, which evolve only parts of solutions predefined by researchers, RePower constructs comprehensive solutions from the ground up in its first stage. Moreover, RePower's integration with physical equipment allows it to control devices and acquire data, enhancing its capability to automate the research process, including obtaining experimental data from measurement equipment, exploring methods using existing data, evolving algorithms for current methods, and providing the code and errors during the whole evolutionary process. By leveraging multiple feedback loops, RePower supports human researchers throughout the entire research cycle, from initial algorithm design to final result improvement.

To evaluate RePower's effectiveness, we validated its performance on three key tasks in power systems: parameter prediction, power optimization, and state estimation. These tasks represent typical data-driven, supervised learning challenges commonly encountered in the field. Additional evaluations of more complex tasks and comparisons with other LLM-based evolution frameworks are detailed in the [supplemental information](#), demonstrating RePower's adaptability and superiority to complex scenarios. The results demonstrate RePower's strong potential as a research assistant in power systems. RePower outperforms existing methods in parameter prediction and shows continuous improvement on tasks without established benchmarks. Compared to the single-stage FunSearch method, RePower's dual-stage approach significantly enhances overall performance. Notably, for tasks with unknown answers, RePower's ability to autonomously refine results mirrors human-like exploration, underscoring its capability to navigate and excel in uncharted research domains.

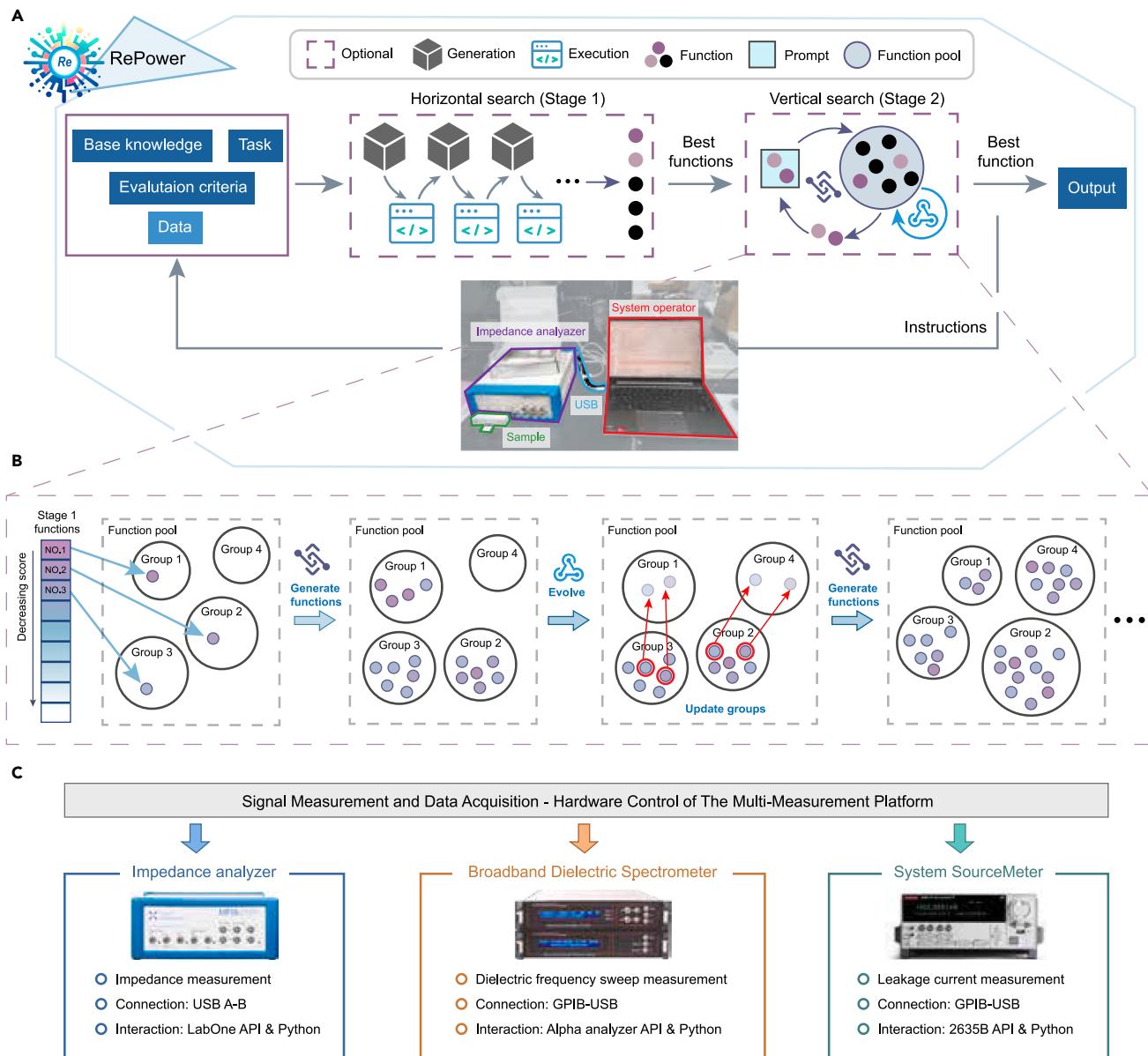
## RESULTS

### Structure of RePower

RePower is a reflection-evolution system designed to automate task completion by integrating multiple components, as illustrated in [Figure 1](#). As depicted in [Figure 1A](#), RePower begins the research process by collecting data and retrieving information directly from physical devices, seamlessly integrating the entire scientific research workflow. The system's final outputs can also be transmitted back to these devices for further testing and validation. For a detailed description of each component's configuration and design, please refer to the [methods](#).

### Preparation

To enable RePower to perform a specific research task, three key elements must be prepared in addition to the data: (1) base knowledge, (2) task descriptions, and (3) evaluation criteria. Base knowledge provides task-independent implementation steps and outlines specific requirements necessary for



**Figure 1. Structure of RePower**

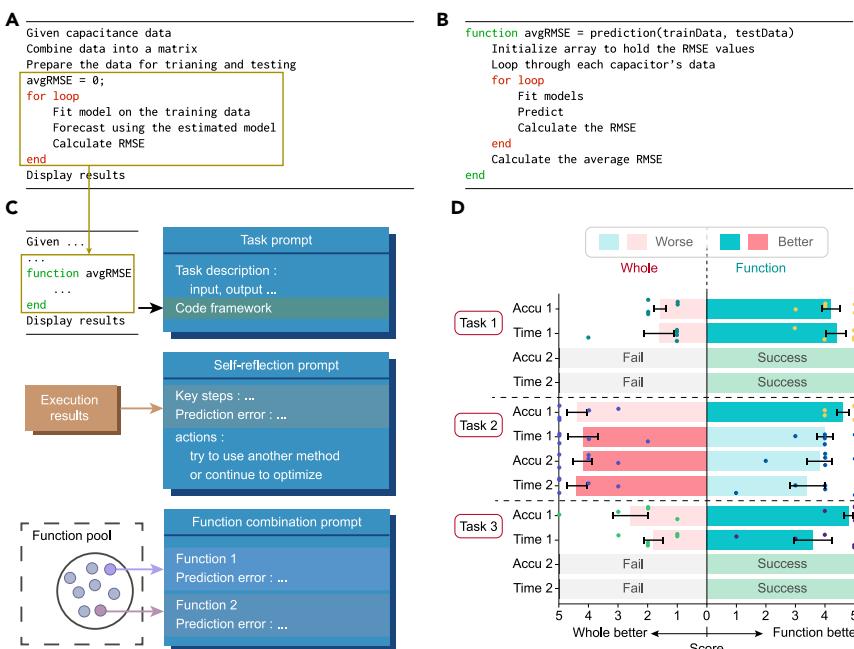
(A) Workflow of RePower. The process begins with the preparation of base knowledge and evaluation criteria. If iterative evolution is selected, then the system undergoes a two-stage search process rather than directly outputting results. In stage 1, the LLM explores different methods using a self-reflection mechanism.<sup>24</sup> The best methods or functions identified in stage 1 are then sent to stage 2 for further refinement if needed. Stage 2 uses extensive generation and evolution to optimize the function, producing the final output. In each iteration, the LLM generates a new function based on feedback.

(B) Evolution process of the function pool. Functions generated in stage 1 are ranked by performance, with top-performing functions entering function groups within the function pool. Each evolution round consists of two steps: sample generation and pool evolution. Samples are generated by merging the best and worst functions from a randomly selected group and added to a group with similar performance. Once the function pool reaches its maximum sample size, two functions are randomly selected from the highest-performing groups to replace functions in other groups, continuing the evolution process until the maximum number of rounds is reached.

(C) Measurement devices for data acquisition. Devices such as the impedance analyzer, broadband dielectric spectrometer, and system sourcemeter are used for data collection. The system can be extended to other API-based instruments, including phasor measurement units in power grids, allowing flexible integration of additional measurement tools.

conducting the research. Task descriptions detail the scientific problem to be addressed, including the task's background, expected inputs and outputs, and the data source. Evaluation

criteria are used to assess the rationality and accuracy of the LLM's output, including the execution of code steps and the correctness of the results obtained.



### Reflection-evolution strategy

The reflection-evolution strategy is a two-stage comprehensive search method, as illustrated in [Figure 1](#). Upon receiving a task, the LLM iteratively generates solution algorithms with executable codes through a self-reflection process.<sup>24</sup> These algorithms are evaluated and sequentially improved based on predefined criteria, typically optimizing the loss for data-driven tasks, resulting in a diverse set of solution methods. To address coding errors introduced by the LLM, an error check-and-correct agent is integrated into the framework to automatically resolve any issues (see “check agent” under [Methods](#) for the detailed design).

In the second stage, the algorithms generated in the first stage are refined further. One or more methods may proceed to this stage for additional enhancement. Using a function pool, this stage applies an islands evolution strategy<sup>14,25,26</sup> to evolve the algorithms ([Figure 1B](#)). Each evolution round involves generating samples and updating the function pool. Simultaneously, the prompt guiding the LLM’s algorithm evolution is refined by merging the best and worst function to improve the search for enhanced solutions. After completing a set number of evolution rounds, the best-performing function from the pool is selected as the final output.

### LLM selection

In this study, we utilized GPT-3.5-turbo-0125,<sup>27</sup> a leading LLM developed by OpenAI, as the agent for both stages of our framework without any additional fine-tuning.<sup>28</sup> Our experiments indicate that the overall effectiveness of the reflection-evolution strategy is not significantly influenced by the specific version of the LLM, suggesting that our framework could be adapted to other LLMs accessible via application programming interface (API). However, the outcomes of individual iterative steps may vary depending on the semantic understanding and generation capabilities of the LLM used. For a detailed comparison with other LLMs, including GPT-4o,<sup>29</sup> please refer to [Note S3](#).

### Figure 2. Characterization of the prompt and generation

(A) Whole-code generation. This example demonstrates MATLAB code generation, though the approach is not limited to this type of code.

(B) Function-only generations. In this approach, the original data and error calculation functions are defined externally, focusing solely on the function generation.

(C) Prompt construction. Task prompt is the combination of task description and the code framework, which can be built based on the complete code generated by the LLM.

(D) Comparison of whole code and function only. “Accu 1” represents the generation accuracy in the first stage, while “Accu 2” denotes the accuracy in the second stage. The same applies to “Time 1”. The data points represent the scores from five experiments, while the bars represent the average scores. The bar chart for some data uses “Success” and “Fail” as labels, where the former means that 10 rounds of iterative search can be completed within the specified time, while the latter cannot. The error bars represent standard error of the mean across five independent tests. For details on the scoring rules, see [Note S8](#).

### Generation patterns

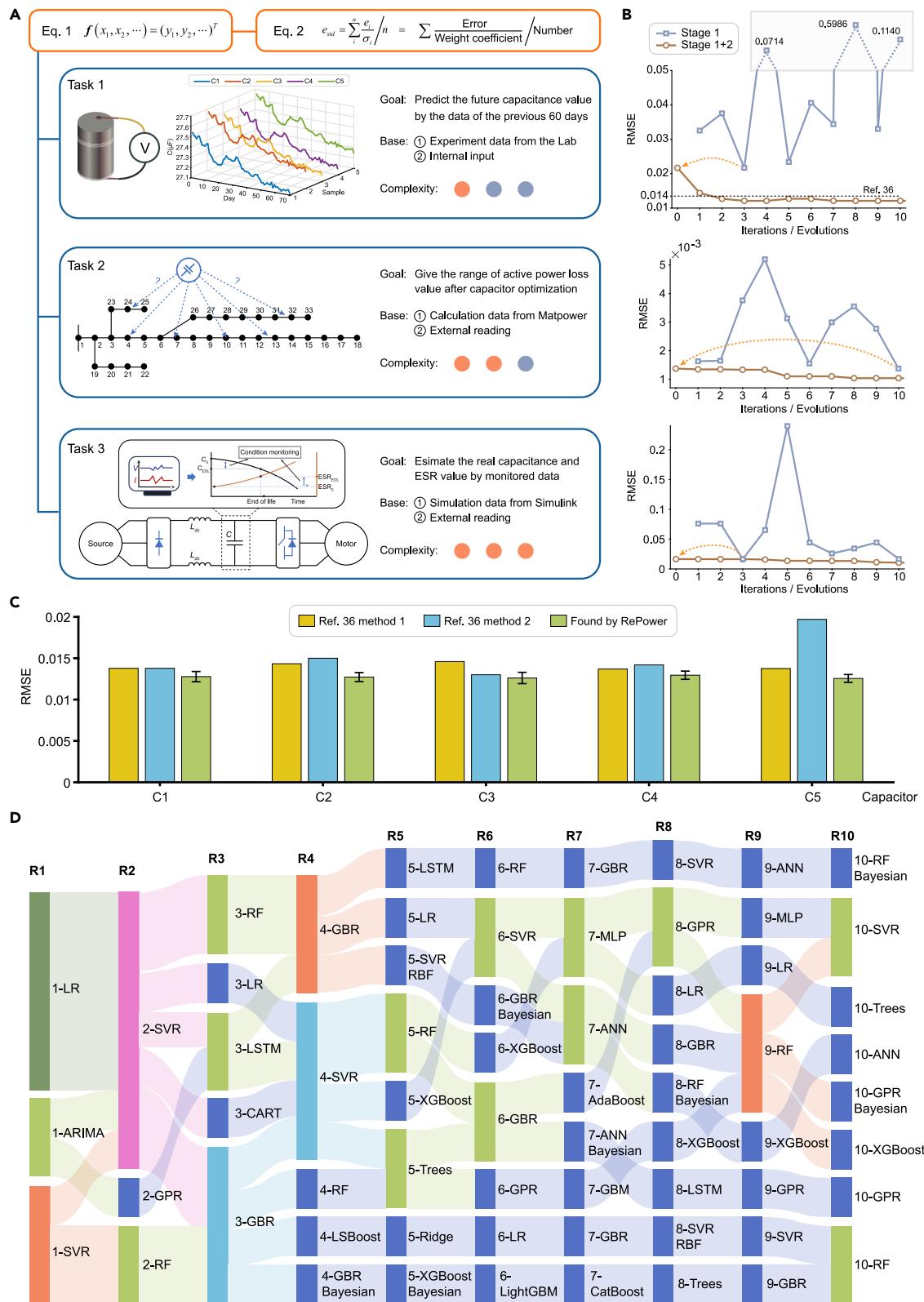
The LLM’s search space is determined by its generation pattern. We designed two patterns: the whole pattern and the function-only pattern, illustrated in [Figures 2A](#) and [2B](#), respectively. The whole pattern involves the LLM generating all code components, including raw data acquisition, data preprocessing, and other relevant steps. In contrast, the function-only pattern restricts the LLM to generating only key functions at crucial stages. Our testing found the function-only pattern to be more effective.

We identified two types of tasks: those with existing solutions and those without. For tasks with existing solutions, researchers can provide a code framework to guide the LLM’s generation process. For tasks without existing solutions, the LLM initially generates a complete code framework based on the natural language description of the problem. Key functions are then extracted and generated separately to avoid redundant code. For example, the generation framework shown in [Figure 2C](#), where essential functions like the “for” loop for prediction and root-mean-square error (RMSE) calculation are abstracted out, forming the streamlined function structure in [Figure 2C](#), task prompt.

### Prompt design

The RePower framework uses distinct prompts across its two stages. In the first stage, two prompts are employed: the task prompt and the self-reflection prompt. The task prompt combines the task description with the code framework ([Figure 2C](#)), guiding the LLM in generating initial solutions. The self-reflection prompt incorporates the evaluation of the LLM’s output and feedback on the next action<sup>30</sup> without including specific details about the function itself.

In the second stage, the prompt is derived from the function pool. This process involves selecting two functions from a randomly chosen group, typically the best- and worst-performing functions, along with their evaluation metrics, such as



(legend on next page)

prediction errors. These components form the function combination prompt (Figure 2C). Both the self-reflection and function combination prompts act as feedback from execution, with distinct prompt generation rules tailored to each stage of the framework.<sup>31</sup>

#### Interaction platform

Our preset interaction platform consists of an impedance analyzer (Zurich Instruments MFIA [medium frequency impedance analyzer]),<sup>32</sup> a sample, and a computer (Figure 1A). We use the LabOne API,<sup>33</sup> a cross-platform tool that enables data acquisition, hardware control, and signal processing, to facilitate communication between RePower and the instruments. Once the communication interface is established, other devices can also be controlled using similar configurations, as shown in Figure 1C. These components form a multi-measurement platform that allows RePower to autonomously conduct corresponding tests in the laboratory. For detailed design and operation of the physical setup, refer to “physical operation” under Methods.

#### Power system research tasks

Many power system research tasks can be abstracted into a mathematical problem (Figure 3A); for example, load forecasting uses historical values to predict future values, satisfying this formula.<sup>34</sup> These tasks involve finding a mapping function that links given inputs to desired outputs, with the primary challenge being to determine this function. While the specific inputs and outputs vary (Table S1), influencing the complexity of each task, the underlying structure remains consistent.

We applied RePower to three typical data-driven tasks in power systems, identified through keyword frequency analysis of a large body of literature (Figure S11). The tasks include parameter prediction, power optimization, and state estimation. The final results of these tasks are presented in Figure S2. Additional results from other tasks can be found in Figure S12.

#### Parameter prediction

Parameter prediction is a common challenge in power systems, such as load forecasting. In this study, we focus on predicting capacitance values as a representative task that integrates both software and hardware aspects. Accurate capacitance prediction is crucial for aging monitoring and fault detection, as a 5% decrease in capacitance indicates the end of a capacitor’s lifespan.<sup>35</sup> We use aging data from a prior study, in which five capacitors were monitored for over 70 days using the ZCtek ZC2811D LCR (inductance capacitance resistance) meter.<sup>36</sup> The capacitor is a biaxially oriented polypropylene capacitor modified with high-dielectric-constant nanoparticles with an initial capacitance close to 27.6 µF. It was measured after being placed in an environment of 22°C and 30% relative humidity for a fixed period of time. The aging curves of these five capacitors are

shown in Figure 3A, task 1. The benchmark task is a point prediction, which requires predicting the capacitance values for the next 10 days based on data collected over the first 60 days, and the effectiveness of the prediction method is evaluated by calculating the RMSE for the last 10 days. The training set and test set are autonomously partitioned by the LLM. In subsequent tests, the LLM will mostly adopt an 80/20 split, meaning the first four capacitors are used as the training set, and the last capacitor is used as the test set.

#### Power optimization

Optimization problems are essential across various stages of power transmission and distribution and are critical for power planning designers. We selected a reactive power compensation optimization problem for RePower to handle<sup>37</sup> As we focus on data-driven tasks, the required data are pre-calculated using the particle swarm optimization algorithm and MATPOWER (Matlab-based power system simulation package).<sup>38</sup> The goal is to determine an optimal indicator, such as the maximum active power loss, given a set of optimization parameters. In this optimization problem, it is necessary to reasonably perform capacitor shunt and series compensation in a 33-node system to minimize active power loss. The objective is to minimize active power loss with a total of 9 variables including the initial series compensation point location, the length of the series compensation point’s effect, the series compensation resistance value, the initial shunt compensation point location 1, the length of the shunt compensation point’s effect 1, the shunt compensation point capacity size 1, the initial shunt compensation point location 2, the length of the shunt compensation point’s effect 2, and the shunt compensation point capacity size 2. The constraints include the power flow equation constraints, the maximum series compensation resistance limit, and the maximum shunt compensation capacity limit. The calculation of active power loss is the sum of all branch losses in the 33-node system. Finally, the maximum and minimum values of active power loss need to be calculated, and the evaluation of the method’s effectiveness is the RMSE of active power loss.

#### State estimation

State estimation is a foundational task in electrical engineering.<sup>39</sup> In this study, we focus on estimating the capacitance and equivalent series resistance (ESR) of the DC (direct current)-link capacitor<sup>40</sup> in power electronic converters, a prerequisite task for capacitance prediction.<sup>41,42</sup> Using Simulink, we simulate the operating voltage and current of the DC-link capacitor under various conditions (Figure S5). The task aims to estimate the capacitance value and ESR of the DC-link capacitor using voltage and current data obtained from simulation over 101 time units, where the effectiveness of the method is evaluated by the RMSE of these two parameters. In this context, we model

---

#### Figure 3. Task profile and evolution results

- (A) Task profile. Three tasks are outlined: the goal (expected results for the research task), base (key steps before starting the system), and complexity (assessment of task difficulty).
- (B) Performance of the two stages. The first stage iterates 10 times, selecting the optimal function as the initial value for the second-stage evolutions.
- (C) The comparison between the final iteration result of task 1 and the existing solution. The abscissa represents different capacitors, while the ordinate represents the predicted error of this method on this capacitor. The error bars represent standard error of the mean in five tests.
- (D) Stage 1 method searching by the LLM. The statistics here are for three tasks described in the main text. The names of these methods were described to users by the LLM, but in the actual code it wrote, the LLM may not have used the corresponding methods, resulting in a certain degree of information bias. For specific methods in abbreviation, please refer to Table S4.

the capacitor using an ideal capacitor in series with a resistor. To simulate the decay process of the capacitor, we set the resistance value to vary between 0.05 and  $0.1\ \Omega$  and the capacitance value to vary between 2 and  $1.6\text{ mF}$ . We continue to employ point prediction, utilizing a total of 202 voltage and current data points to predict the capacitor and ESR values. The training set and test set are still autonomously divided by the LLM.

Before conducting a complete test, we assessed the performance of the three tasks using the two previously described generation patterns: whole generation and function-only generation. As shown in Figure 2D, the function-only generation pattern outperforms whole generation in tasks 1 and 3 but slightly underperforms in task 2. In task 1, which involves direct input of raw data (Figure 3A), generating complete code leads to repeated data copying by the LLM, causing errors and reduced accuracy. In task 3, which requires reading data from external sources, the complexity of data type conversion significantly decreases accuracy during the first stage. As a result, the second-stage accuracy for tasks 1 and 3 is not displayed in Figure 2D because the first stage could not generate the required 10 functions due to the extended logical chain and increased error probability in complete generation. GPT-3.5 struggles to generate all functions within the allowed time and iteration limits. While both the whole and function-only generation patterns can be used with an LLM like GPT-3.5, the function-only pattern generally performs better for tasks involving complex data types or direct data inputs in RePower due to its shorter reasoning steps.

### Performance on the three tasks

We evaluated the accuracy of calculations using normalized RMSE for multiple outputs (Figure 3A, weighted average error) and standard RMSE for single outputs. As shown in Figure 3B, the results of the first stage did not improve with increased search iterations, which aligns with our expectations. In the first stage, the objective is not to find the best methods but to generate diverse solutions that can guide the evolutionary process in the second stage. After only 10 rounds of evolution in stage 2, RePower's performance on task 1 surpassed the benchmark method from Zhang et al.<sup>36</sup> (smoothing spine with  $\text{RMSE} \approx 0.014$ , which uses a piecewise low-order polynomial fitting method to make the curve smoother while maintaining overall stability and effectively preventing overfitting<sup>43</sup>), which used the same dataset and claimed to be the best. This highlights RePower's capability to enhance and evolve methods through automated processes. The line chart in Figure 3B shows a continuous error reduction with increased iterations in the second stage until stabilization, indicating successful ongoing algorithm evolution under RePower. Although similar trends were observed across tasks, task 1 exhibited a notably faster error decline. After 20 rounds of the two-stage evolutionary process, task 1's RMSE decreased from 0.0144 to 0.0119, representing an average reduction of 9.29% compared to Zhang et al.<sup>36</sup> (Figure 3C). The method evolved from initial linear regression to compound linear regression. For task 2, the RMSE decreased from 0.00137 to 0.000825, with the method evolving from a simple decision tree ensemble to one incorporating Bayesian optimization and feature dimension reduction. Task 3's RMSE dropped from 0.0165 to 0.00971, transitioning from support vector

regression (SVR) to SVR with added polynomial features. All tasks achieved favorable results, demonstrating the effectiveness of RePower's automatic solution evolution.

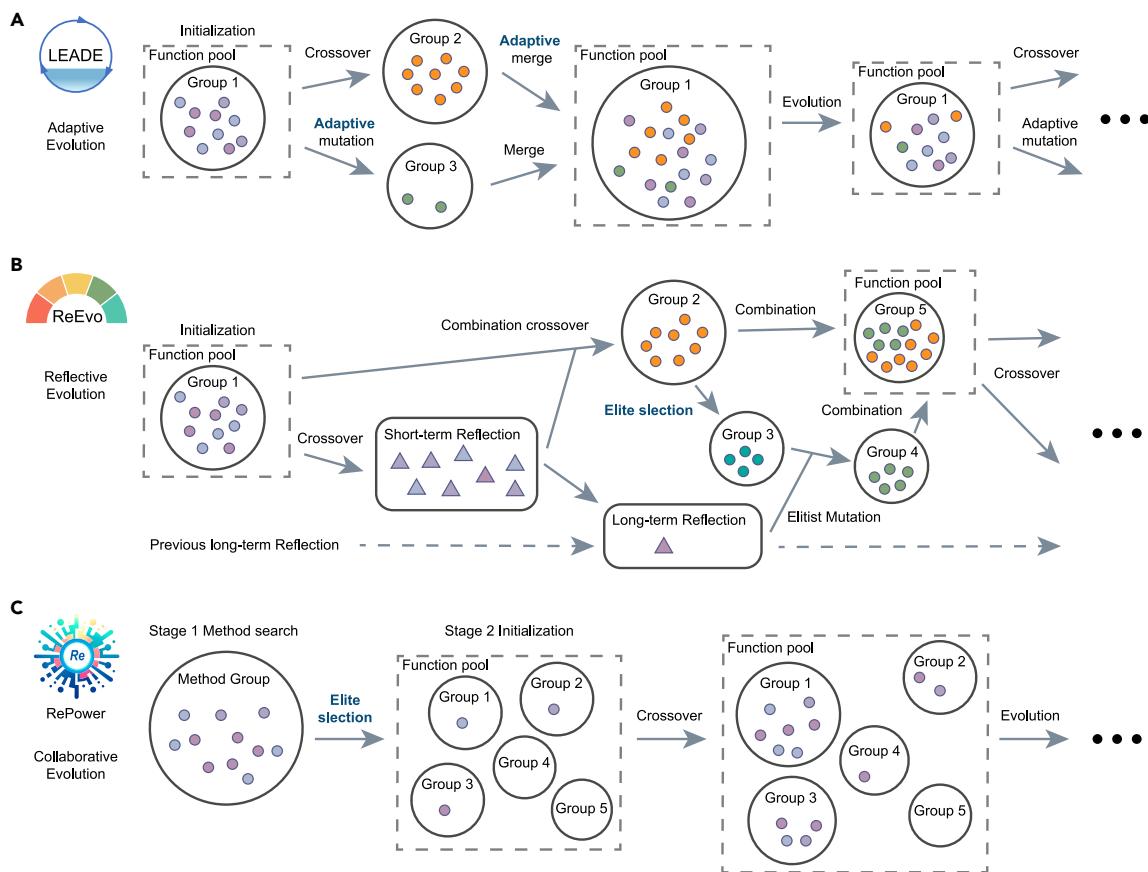
In task 1, the overall trend of capacitance value attenuation is linear but exhibits fluctuations over different periods (Figure 3A). In the process of algorithm evolution, LLMs initially attempt polynomial fitting from low order to high order, followed by a comparison with the SVR method. Subsequently, hyperparameter tuning is applied to SVR to enhance prediction accuracy. The final evolved solution combines linear and SVR approaches, capturing the general linear decline with added nonlinear adjustments for observed disturbances (Figure S2A). Task 2 involves nine optimization variable parameters. During evolution, the LLM employs principal-component analysis (PCA) for dimensionality reduction and feature selection, followed by Bayesian optimization for hyperparameter optimization and regression trees for prediction (Figure S2B). Since the nine-dimensional raw data contain fixed elements that do not influence the final outcome, dimensionality reduction improves performance by removing irrelevant components. Task 3 deals with time-series data of voltage and current, requiring feature extraction to reveal the relationships between state parameters, capacitance values, and ESR. The final solution algorithm developed by RePower first extracts features using a cubic polynomial, which are then fed into a fully connected feedforward neural network for training and prediction. The hyperparameters of the neural network were not adjusted or optimized (Figure S2C).

### Comparison with other LLM-based evolutionary frameworks

To further validate RePower's effectiveness, we compared it with two other evolutionary frameworks similar to RePower, LEADE (LLM-enhanced adaptive evolutionary search)<sup>44</sup> and ReEvo (Reflective Evolution).<sup>45</sup> Comparative diagrams of these frameworks and RePower are provided in Figure 4.

LEADE begins by initializing 30 functions in response to a given task prompt, selecting 10 of these to form the initial function group. From this group, adaptive crossover and mutation operations are applied to generate new functions, which are then merged with the original set, expanding the function group. This completes one cycle of the evolutionary process. ReEvo follows the same initialization process as LEADE but introduces short-term and long-term reflections during the crossover and mutation phases. Unlike LEADE, ReEvo replaces the original function group with the newly generated one rather than merging the two. In contrast, RePower employs an initial reflection phase to explore diverse methods, followed by a group evolutionary search to iteratively optimize these methods, functioning as a collaborative evolutionary system.

The performance of the three frameworks across the first three tasks is presented in Figure 5. It reveals that LEADE outperformed RePower only in task 1, while RePower identified superior methods for the remaining two tasks. This disparity arises because task 1 is relatively simple, involving only a single changing parameter, which makes heuristic search strategies effective in finding better predictive methods. However, this approach becomes less effective in more complex problems, as reflected in tasks 2 and 3, where LEADE and ReEvo exhibit plateaus, indicating that iterative searches reached stagnation.



**Figure 4. Workflow diagram comparison of different evolutionary frameworks**

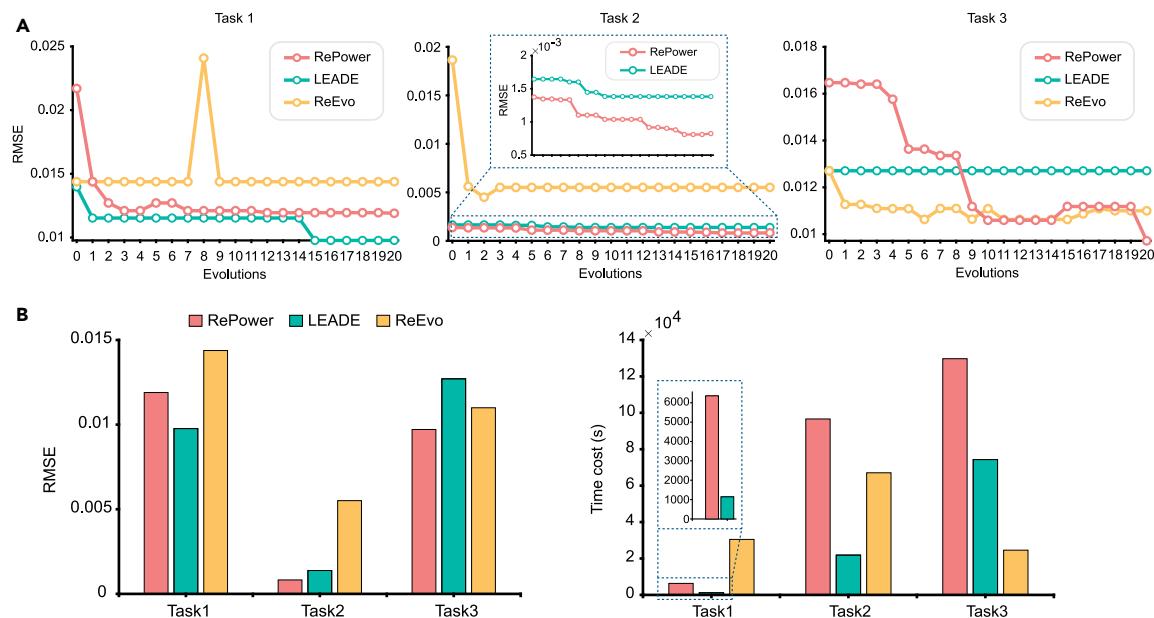
(A) Adaptive evolution of LEADE. It is characterized by adaptively performing crossover and mutation processes according to the fitness of the function. (B) Reflective evolution (ReEvo). Its characteristic is to add long-term and short-term reflection of speech in the process of crossover and mutation, and this reflection is also a kind of crossover. (C) Collaborative evolution of RePower. Its characteristic is the collaboration of a two-stage search process, where the first stage involves method search, and the second stage entails group evolutionary search.

In many cases, LEADE and ReEvo tend to converge on linear or polynomial regression, making incremental improvements without evolving to more sophisticated methods, contributing to their underperformance in challenging tasks. It is noteworthy that RePower generally requires a longer search time owing to its more complex reflection and evolution strategy. Yet, it is worthwhile to trade long thinking time for the quality of the solution.

#### Performance factors and explorations of RePower

Given RePower's numerous operational steps, we focused on factors affecting its core evolutionary performance. Figure 6 presents the impact of several key explorations. Our results show that both stages of RePower's self-improvement process are critical and cannot be substituted for one another. The first stage alone does not achieve continuous improvement (Figure 3B), while the second stage alone, though offering partial evolution (Figure 6A), performs worse in prediction error (53.40% vs. 38.27%) and has a higher generation error rate compared to the combined two-stage approach (Figure 6B). Generation error rate is defined as the proportion of incorrectly executed samples

out of the total generated samples (see Note S7), demonstrating that RePower outperforms FunSearch's single-stage evolution. Without the first stage, there is insufficient diversity in the initial functions, often causing the LLM to default to linear regression or occasionally other methods, like random forests (Figure 6C). This lack of diversity limits the LLM's ability to direct the evolution process, making subsequent searches more random and reducing overall performance. In the absence of robust initial function screening, the average generation error rate increases, resulting in suboptimal final outcomes. Given the significant influence of initial values from stage 1 on stage 2's performance, we further examined how varying initial values affects stage 2's output (Figure 6D). Different starting values produced different results, with the best function from stage 1 not always leading to the optimal outcome due to the risk of getting trapped in local optima. To mitigate this limitation, we introduced multiple initial functions (denoted as  $k$ ) from stage 1 into stage 2 for iteration. The results indicated that the best performance was achieved with  $k = 3$  (Figure 6E). A smaller  $k$  reduces search diversity, while a larger  $k$  dilutes the optimal function by introducing many non-beneficial functions, potentially compromising overall performance.



**Figure 5. Task results comparison using different evolutionary frameworks**

(A) The evolutionary process comparison of the three frameworks. The result of RePower is the second-stage evolutionary process that connects the first stage.  
(B) The final result comparison of the three frameworks.

## DISCUSSION

We demonstrated RePower’s effectiveness as a research designer capable of conceptualizing new algorithms and continuously refining them. Its strong adaptability to physical interfaces (Figure 1C) makes RePower highly suitable for client-side deployment. Compared to other evolutionary frameworks, RePower’s approach of evolving multiple function groups enhances individual diversity, optimizing overall performance (Figure 5). The entire process is highly automated, with task completion times significantly shorter than human efforts—less than half the time—while maintaining reasonable costs (Figure S15). RePower’s efficiency and automation highlight its potential to revolutionize research processes in power systems and beyond.

Despite its strengths, RePower has some limitations. Due to constraints in laboratory equipment, interactions with more complex devices for enhanced designs remain incomplete but offer opportunities for future improvement. We also aim to develop a unified generation framework compatible with the LLM environment, designed to handle diverse data operations to produce outputs. The current framework, primarily focused on predictive functions, may be inadequate, as it requires identifying commonalities across various methods. Additionally, while testing large models typically involves extensive datasets with explicit answers,<sup>46</sup> our evaluation relied on a limited number of smaller cases with unknown or open-ended outcomes. Despite these limitations, RePower demonstrates significant potential in designing and evolving solutions, highlighting the need for further exploration and refinement.

RePower’s reflection-evolution strategy effectively broadens the search space while efficiently completing tasks. Simply generating numerous function samples and screening them in the second stage, without method selection in the first stage, would not

achieve optimal results. This is because an LLM without fine-tuning or contextual information tends to generate algorithms based on the most probable outcomes from its pre-training, often defaulting to basic methods like linear regression, thereby missing opportunities to explore diverse approaches. Similarly, without the in-depth reflection provided in the second stage, many methods remain surface level and fail to reach their full potential. RePower’s success lies in its reflection-evolution strategy, which integrates horizontal and vertical search, combines long-term and short-term memory, and incorporates human-like feedback with execution. This comprehensive approach enables RePower to navigate complex tasks with depth and adaptability. Looking ahead, we anticipate further improvements in this automated system, enhancing its role as a versatile research assistant across various fields. RePower holds the promise of advancing scientific exploration, providing novel insights, and supporting human researchers in unprecedented ways.

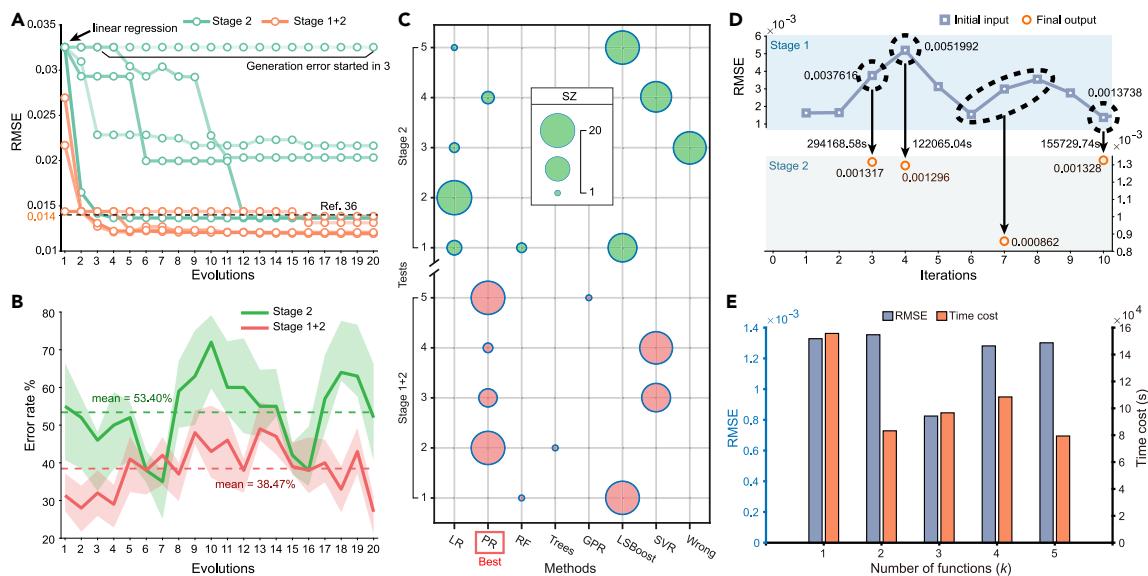
## METHODS

### Datasets

This study did not use any open-source datasets in the main text; instead, we independently constructed datasets tailored to each specific task. The capacitance data were directly measured in our laboratory, as detailed in Zhang et al.<sup>36</sup> Data for the other two tasks were generated using MATLAB simulations (Figure S5). For additional tasks presented in the supplemental information, we employed datasets from the UCI (University of California Irvine) Machine Learning Repository.<sup>47</sup>

### LLM

The LLM serves as the core of RePower’s innovation, with GPT-3.5-turbo-0125 used for generating task-specific instructions



**Figure 6. Exploration of the RePower's performance factors**

(A) RMSE comparison. This compares RMSE of using only stage 2 versus both stage 1 and 2 of task 1. RMSE represents the normalized standard error when there are multiple outputs.

(B) Average error rate comparison. This compares the average error rate between using only stage 2 and both stages 1 and 2 of task 1. The width of the band represents the standard error of the mean.

(C) Method comparison. Here, we compare the methods that emerged during 20 evolutionary processes of task 1, using only stage 2 and both stage 1 and 2, respectively, across 5 experiments. SZ represents size, indicating that the frequency is presented in the form of size. The “PR” method is the best approach suitable for this task (found by our system), whereas the “Wrong” method implies that there is a flaw in the computational logic.

(D) Initial value selection performance. This compares the performance of initial value selection in the first stage of task 2. Results are based on selecting functions enclosed by black dashed circles to enter the second stage of iteration.

(E) Function number comparison. This compares the performance of different quantities of functions transitioning from stage 1 to stage 2 of task 2.

and algorithm codes. A comparative analysis between GPT-3.5 and the more advanced GPT-4o<sup>29</sup> revealed only marginal improvements in individual results with GPT-4o without a significant impact on overall self-improvement performance. Both models also demonstrated similar error types and frequencies (Figure S10). Based on these findings, we selected GPT-3.5 as the primary tool for our system.

### Prompt generation

In stage 1, apart from the manually provided task prompts, each self-reflection prompt was automatically generated based on preset scenarios involving simple conditional statements.<sup>48</sup> If execution was correct, then the response was constructed using the return value, typically highlighting a prediction error and suggesting alternative methods for subsequent steps. In cases of errors or timeouts, the prompt included the error code and message to attempt a fix, with the check agent deployed to address these issues. If the error could not be resolved (e.g., the LLM failed to understand the error and made arbitrary changes), the LLM was instructed to regenerate a new solution to ensure the iteration continued without interruption.

Stage 2 prompts were generated using a combination of the best and worst functions from a randomly selected group within the function pool (Figure 2C, function combination prompt). This approach, inspired by evolutionary algorithms, was designed to help the LLM identify functional differences and drive continuous improvement. Experimental results demonstrated that this com-

bination was more effective in finding breakthroughs (Figure S3A). Combining the best and worst functions enhanced evolutionary outcomes by maximizing mutation potential and avoiding local optima. Throughout the process, we prioritized group optimality over individual performance, allowing for fluctuations that ensured significant deviations in the evolutionary path and enhanced the overall refinement process.

### Check agent

In the first stage of iteration, our goal was for the LLM to maintain contextual memory to correctly generate various methods for sequential execution. However, the LLM-generated code frequently contained errors that could disrupt the program. To address this, we developed an LLM-based check agent to identify and correct code errors (Figure S1). This agent operates as a proxy for the main chat, focusing specifically on error correction. When an error occurs in the main chat, the error code and message are transferred to the check agent, which guides the LLM in repairing the issue. Once corrected, the revised code and updated return value are sent back to the main chat. If the errors cannot be resolved despite multiple attempts, the main chat's context is reset, and the process is regenerated to ensure continuous iteration.

### Evolution strategy

The evolution strategy is central to the second stage of RePower's iterative process. The first stage involves a horizontal

**Algorithm 1. Evolution strategy**

```
Initialize individuals, groups:  $f, g = \{ f \}$ 
Initialize function pool  $P = \{ g \}$ 
while  $i < \text{max evolutions}$  do
    while  $j < \text{max samples}$  do
        Randomly select  $g'$  in  $P$ 
        Construct prompt  $s$  using  $f$  in  $g'$ 
        Generate individuals  $f'$  using prompt  $s$ 
        if  $f'$  pass
            Append  $f'$  to  $P$ 
        end if
        Increment  $j$ 
    end while
    Evolve  $P$ 
    Increment  $i$ 
end while
return
```

search conducted by the LLM using a self-reflection strategy. Although these operations could be performed using existing interfaces like MatGPT<sup>49</sup> and Assistant API,<sup>50</sup> we opted to use our own interfaces to facilitate system processing, feedback integration, and strategy improvement. In stage 2, we employed an island-based genetic algorithm for evolutionary search.<sup>51–53</sup> This strategy comprises three hierarchical components: the function pool, groups, and individuals.

(1) The function pool is the largest environment, akin to a society, where all correct output samples generated by the LLM reside as individuals. It serves as the primary space for evolution. (2) Groups are smaller subsets within the function pool, containing function samples that directly influence evolution. When a correct sample is added to the function pool, it is assigned to a group with similar properties, such as average error. Each group marks its best and worst individuals for future prompt construction. Adding new individuals updates the group's performance metrics, including the average error. (3) Individuals are the smallest units in the function pool, each represented by MATLAB code, output information (such as prediction error), and other details (e.g., code length). The terms "individual" and "function" are used interchangeably due to minimal differences between them.

The evolution process unfolds in several steps (Figure 1B). Initially, the function pool is seeded with the best-performing functions from the first stage, which are then sorted into groups. Evolution begins once sufficient samples are generated in each round. Functions selected for evolution are the top  $k$  from the first stage, and  $k + n$  function groups are formed, where  $n$  accounts for additional groups that maintain diversity and store outlier functions during early evolution (Figure 1B shows the effect with  $n = 1$ ). Subsequent experiments identified optimal performance with  $k = 3$  and  $n = 2$  (Figure S3B). During sample generation, correctly generated functions are added to the group with the closest error value. After all samples are generated, the function pool undergoes evolution. Functions are randomly selected from the top-performing groups to replace those in other groups, updating group errors and identifying the best and worst individuals.

To prevent degradation due to an excess of individuals in a single group (where inferior functions could outnumber and drive

out superior ones),<sup>52,54</sup> any group exceeding 10 individuals randomly retains 9 plus the current optimal function. This approach ensures that each group maintains 10 individuals, preserving diversity and stability in the evolutionary process.

**Physical operation**

The physical operations in this study are confined to the direct control of electronic test equipment for data acquisition, excluding any mechanical motion components. As shown in Figure 1C, we developed a multi-measurement platform capable of controlling three devices: the MFIA impedance analyzer,<sup>32</sup> the Alpha-A broadband dielectric spectrometer,<sup>55</sup> and the Keithley system sourcemeter.<sup>56</sup> The MFIA impedance analyzer connects directly to the computer via USB (universal serial bus) (A-B), while the Alpha-A spectrometer and the Keithley sourcemeter communicate through the National Instruments GPIB (general purpose interface bus)-USB-HS (high speed) interface.<sup>57</sup> To ensure safety, particularly in light of potential LLM hallucinations,<sup>12,58</sup> the LLM generates only control instructions, which are then executed by pre-written API programs. This approach simplifies the process and ensures safety. For example, when using the impedance analyzer, RePower generates control instructions that are sent to the device via its API<sup>33</sup> to conduct the test. The device then returns the data for the specified parameters, which is subsequently used in RePower's two-stage iterative self-improvement process to further refine the solution for the specific task (Figures S7–S9).

**Extended skeleton**

To improve the efficiency of the LLM, we developed an extended skeleton framework that abstracts repetitive steps commonly found in our method, such as data extraction and error calculation. This framework focuses the LLM on generating only the key steps of each process, minimizing redundant operations. By abstracting these common steps, the framework enables the LLM to employ heuristic thinking to identify better alternative approaches and incrementally refine solutions. Additionally, this framework helps prevent the LLM from generating solutions that, while producing correct results, do not align with the task's specific requirements. An example of a simplified framework is shown in Figure 2C, illustrating how structured guidance enhances logical coherence in the LLM's output.

**RESOURCE AVAILABILITY****Lead contact**

Requests for further information, resources, and reagents should be directed to and will be fulfilled by the lead contact, Zhi-Min Dang ([dangzm@tsinghua.edu.cn](mailto:dangzm@tsinghua.edu.cn)).

**Materials availability**

This study did not generate new materials or reagents.

**Data and code availability**

- All original code with experiment data has been deposited at Zenodo<sup>59</sup> (<https://doi.org/10.5281/zenodo.14216501>) and is publicly available as of the date of publication.
- ChatGPT conversation generation examples of the 6 tasks mentioned in this study have been deposited at Zenodo<sup>60</sup> at <https://doi.org/10.5281/zenodo.14219850> and are publicly available as of the date of publication.

- Any additional information required to reanalyze the data reported in this paper is available from the [lead contact](#) upon request.

## ACKNOWLEDGMENTS

This work was financially supported by National Key Research and Development Program of China (2021YFB2401504)

## AUTHOR CONTRIBUTIONS

Conceptualization, Z.M.D., Y.-X.L., and M.J.; methodology, M.J.; investigation, Y.-X.L., Y.-X.Z., and M.J.; visualization, Y.-X.L. and Y.-X.Z.; writing – original draft, Y.-X.L.; writing – review & editing, J.W., G.H., and Z.M.D.; funding acquisition, S.-L.Z. and Z.M.D.; project administration, S.L.Z.; supervision, Z.M.D.

## DECLARATION OF INTERESTS

The authors declare no competing interests.

## DECLARATION OF GENERATIVE AI AND AI-ASSISTED TECHNOLOGIES

During the preparation of this work the authors used GPT-4o in order to polish the academic writing. After using this tool, the authors reviewed and edited the content as needed and take full responsibility for the content of the publication.

## SUPPLEMENTAL INFORMATION

Supplemental information can be found online at <https://doi.org/10.1016/j.patter.2025.101211>.

Received: November 23, 2024

Revised: January 27, 2025

Accepted: March 6, 2025

Published: March 31, 2025

## REFERENCES

- Mei, S., Zhang, X., and Cao, M. (2011). *Power Grid Complexity* (Tsinghua University Press, Beijing and Springer-Verlag Berlin Heidelberg).
- Lv, C., Liu, J., and Zhang, Y. (2022). A data-driven method for anomaly detection and aging model parameter estimation of capacitors based on condition monitoring. *Microelectron. Reliab.* 138, 114646. <https://doi.org/10.1016/j.micrel.2022.114646>.
- Liu, Z., Fang, L., Jiang, D., and Qu, R. (2022). A Machine-Learning-Based Fault Diagnosis Method With Adaptive Secondary Sampling for Multiphase Drive Systems. *IEEE Trans. Power Electron.* 37, 8767–8772. <https://doi.org/10.1109/tpe.2022.3153797>.
- Wu, W., Qiu, L., Liu, X., Guo, F., Rodriguez, J., Ma, J., and Fang, Y. (2022). Data-Driven Iterative Learning Predictive Control for Power Converters. *IEEE Trans. Power Electron.* 37, 14028–14033. <https://doi.org/10.1109/tpe.2022.3194518>.
- Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language Models are Few-Shot Learners. Proceedings of the 34th International Conference on Neural Information Processing Systems (NeurIPS). In Curran Associates Inc., H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, eds., pp. 1877–1901.
- Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., Yogatama, D., Bosma, M., Zhou, D., Metzler, D., et al. (2022). Emergent Abilities of Large Language Models. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2206.07682>.
- Dai, D., Sun, Y., Dong, L., Hao, Y., Sui, Z., and Wei, F. (2022). Why Can GPT Learn In-Context? Language Models Secretly Perform Gradient Descent as Meta-Optimizers. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2212.10559>.
- Schäfer, M., Nadi, S., Eghbali, A., and Tip, F. (2023). An Empirical Evaluation of Using Large Language Models for Automated Unit Test Generation. *IEEE Trans. Softw. Eng.* 50, 1–21. <https://doi.org/10.1109/tse.2023.3334955>.
- Kwon, S., Lee, S., Kim, T., Ryu, D., and Baik, J. (2023). Exploring LLM-based Automated Repairing of Ansible Script in Edge-Cloud Infrastructures. *J. Web Eng.* 22, 889–912. <https://doi.org/10.13052/jwe1540-9589.2263>.
- Singhal, K., Azizi, S., Tu, T., Mahdavi, S.S., Wei, J., Chung, H.W., Scales, N., Tanwani, A., Cole-Lewis, H., Pfohl, S., et al. (2023). Large language models encode clinical knowledge. *Nature* 620, 172–180. <https://doi.org/10.1038/s41586-023-06291-2>.
- Wang, Z., Chen, A., Tao, K., Han, Y., and Li, J. (2024). MatGPT: A Vane of Materials Informatics from Past, Present, to Future. *Adv. Mater.* 36, e2306733. <https://doi.org/10.1002/adma.202306733>.
- Boiko, D.A., MacKnight, R., Kline, B., and Gomes, G. (2023). Autonomous chemical research with large language models. *Nature* 624, 570–578. <https://doi.org/10.1038/s41586-023-06792-0>.
- M Bran, A., Cox, S., Schilter, O., Baldassari, C., White, A.D., and Schwaller, P. (2024). Augmenting large language models with chemistry tools. *Nat. Mach. Intell.* 6, 525–535. <https://doi.org/10.1038/s42256-024-00832-8>.
- Romera-Paredes, B., Barekatain, M., Novikov, A., Balog, M., Kumar, M.P., Dupont, E., Ruiz, F.J.R., Ellenberg, J.S., Wang, P., Fawzi, O., et al. (2024). Mathematical discoveries from program search with large language models. *Nature* 625, 468–475. <https://doi.org/10.1038/s41586-023-06924-6>.
- Huang, C., Li, S., Liu, R., Wang, H., and Chen, Y. (2023). Large Foundation Models for Power Systems. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2312.07044>.
- Majumder, S., Dong, L., Doudi, F., Cai, Y., Tian, C., Kalathil, D., Ding, K., Thatte, A.A., Li, N., and Xie, L. (2024). Exploring the capabilities and limitations of large language models in the electric energy sector. *Joule* 8, 1544–1549. <https://doi.org/10.1016/j.joule.2024.05.009>.
- Arnautov, K.V., and Akimov, D.A. (2024). In Application of Large Language Models for Optimization of Electric Power System States. 2024 Conference of Young Researchers in Electrical and Electronic Engineering (ElCon), S. Shaposhnikov, ed. (IEEE), pp. 314–317. <https://doi.org/10.1109/elcon61730.2024.10468377>.
- Yan, Z., and Xu, Y. (2024). Real-Time Optimal Power Flow with Linguistic Stipulations: Integrating GPT-Agent and Deep Reinforcement Learning. *IEEE Trans. Power Syst.* 39, 4747–4750. <https://doi.org/10.1109/tpwrs.2023.3338961>.
- Shahzad, K., Iqbal, S., and Moazam Fraz, M. (2023). Automated Solution Development for Smart Grids: Tapping the Power of Large Language Models. In 2023 17th International Conference on Engineering of Modern Electric Systems (EMES) (IEEE), pp. 1–4. <https://doi.org/10.1109/emes58375.2023.10171681>.
- Ruan, J., Liang, G., Zhao, H., Liu, G., Sun, X., Qiu, J., Xu, Z., Wen, F., and Dong, Z.Y. (2024). Applying Large Language Models to Power Systems: Potential Security Threats. *IEEE Trans. Smart Grid* 15, 3333–3336. <https://doi.org/10.1109/tsg.2024.3373256>.
- Xu, X., Jia, Y., Xu, Y., Xu, Z., Chai, S., and Lai, C.S. (2020). A Multi-Agent Reinforcement Learning-Based Data-Driven Method for Home Energy Management. *IEEE Trans. Smart Grid* 11, 3201–3211. <https://doi.org/10.1109/tsg.2020.2971427>.
- Yuan, Y., Wang, Y., and Wang, Z. (2024). A Data-Driven Framework for Power System Event Type Identification via Safe Semi-Supervised Techniques. *IEEE Trans. Power Syst.* 39, 1460–1471. <https://doi.org/10.1109/tpwrs.2023.3266153>.
- Zhang, Y.-X., Feng, Q.-K., Zhong, S.-L., Pei, J.-Y., Chen, F.-Y., He, G.-N., and Dang, Z.-M. (2021). Digital twin accelerating development of

- metallized film capacitor: Key issues, framework design and prospects. *Energy Rep.* 7, 7704–7715. <https://doi.org/10.1016/j.egyr.2021.10.116>.
24. Peng, B., Galley, M., He, P., Cheng, H., Xie, Y., Hu, Y., Huang, Q., Liden, L., Yu, Z., Chen, W., et al. (2023). Check Your Facts and Try Again: Improving Large Language Models with External Knowledge and Automated Feedback. Preprint at arXiv. <https://doi.org/10.48550/arxiv.2302.12813>.
  25. Tanese, R. (1989). *Distributed Genetic Algorithms for Function Optimization (Deep Blue (University of Michigan))*.
  26. Cantú-Paz, E., and Goldberg, D.E. (1999). On the Scalability of Parallel Genetic Algorithms. *Evol. Comput.* 7, 429–449. <https://doi.org/10.1162/evco.1999.7.4.429>.
  27. Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. (2022). Training language models to follow instructions with human feedback. Proceedings of the 36th International Conference on Neural Information Processing Systems (NeurIPS). In Curran Associates Inc., S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, eds., pp. 27730–27744.
  28. Chen, B., Yi, F., and Varró, D. (2023). Prompting or Fine-tuning? A Comparative Study of Large Language Models for Taxonomy Construction. In 2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C) (IEEE), pp. 588–596. <https://doi.org/10.1109/models-c59198.2023.00097>.
  29. OpenAI (2024). Gpt-4o release. Openai.com. <https://openai.com/index/hello-gpt-4o/>.
  30. Lu, L.-C., Chen, S.-J., Pai, T.-M., Yu, C.-H., Lee, H., and Sun, S.-H. (2024). LLM Discussion: Enhancing the Creativity of Large Language Models via Discussion Framework and Role-Play. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2405.06373>.
  31. Hu, Y., Tang, X., Yang, H., and Zhang, M. (2024). Case-Based or Rule-Based: How Do Transformers Do the Math?. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2402.17709>.
  32. Zurich Instruments AG (2019). MFIA 500 kHz/5 MHz Impedance Analyzer | Zurich Instruments. www.zhinst.com. <https://www.zhinst.com/americas/en/products/mfia-impedance-analyzer>.
  33. Zurich Instruments AG (2023). LabOne API User Manual. docs.zhinst.com. [https://docs.zhinst.com/labone\\_api\\_user\\_manual/index.html](https://docs.zhinst.com/labone_api_user_manual/index.html).
  34. Wang, Q., Li, F., Tang, Y., and Xu, Y. (2019). Integrating Model-Driven and Data-Driven Methods for Power System Frequency Stability Assessment and Control. *IEEE Trans. Power Syst.* 34, 4557–4568. <https://doi.org/10.1109/twrs.2019.2919522>.
  35. Wu, X., Li, K., Tian, R., Yin, H., Yu, T., Cheng, S., and Chen, C. (2024). A Capacitance Estimation Method for DC-Link Capacitors in Railways Based on Pre-Charging Model and Low Sampling Frequency. *IEEE Trans. Power Electron.* 39, 1527–1537. <https://doi.org/10.1109/tpel.2023.3322699>.
  36. Zhang, Y.-X., Feng, Q.-K., Chen, F.-Y., Liu, D.-F., Pei, J.-Y., Zhong, S.-L., Yang, Z., and Dang, Z.-M. (2022). Long-term capacitance variation characteristics, law extraction, single and collaborative prediction of film capacitors at room temperature and humidity. *Microelectron. Reliab.* 139, 114845. <https://doi.org/10.1016/j.micrel.2022.114845>.
  37. Dai, C., Chen, W., Zhu, Y., and Zhang, X. (2009). Seeker Optimization Algorithm for Optimal Reactive Power Dispatch. *IEEE Trans. Power Syst.* 24, 1218–1231. <https://doi.org/10.1109/twrs.2009.2021226>.
  38. Zimmerman, R.D., Murillo-Sánchez, C.E., and Thomas, R.J. (2011). MATPOWER: Steady-State Operations, Planning, and Analysis Tools for Power Systems Research and Education. *IEEE Trans. Power Syst.* 26, 12–19. <https://doi.org/10.1109/TPWRS.2010.2051168>.
  39. Course, K., and Nair, P.B. (2023). State estimation of a physical system with unknown governing equations. *Nature* 622, 261–267. <https://doi.org/10.1038/s41586-023-06574-8>.
  40. Zhao, Z., Davari, P., Lu, W., Wang, H., and Blaabjerg, F. (2021). An Overview of Condition Monitoring Techniques for Capacitors in DC-Link Applications. *IEEE Trans. Power Electron.* 36, 3692–3716. <https://doi.org/10.1109/tpel.2020.3023469>.
  41. McGrew, T., Sysoeva, V., Cheng, C.-H., Miller, C., Scofield, J., and Scott, M.J. (2022). Condition Monitoring of DC-Link Capacitors Using Time-Frequency Analysis and Machine Learning Classification of Conducted EMI. *IEEE Trans. Power Electron.* 37, 12606–12618. <https://doi.org/10.1109/TPEL.2021.3135873>.
  42. Yao, B., Wei, X., Zhang, Y., Correia, P., Wu, R., Song, S., Trintis, I., Wang, H., and Wang, H. (2024). Accelerated Degradation Testing and Failure Mechanism Analysis of Metallized Film Capacitors for AC Filtering. *IEEE Trans. Power Electron.* 39, 6256–6270. <https://doi.org/10.1109/tpe.2024.3360373>.
  43. Gu, C. (2002). Smoothing Spline ANOVA Models (Springer New York, NY). <https://doi.org/10.1007/978-1-4757-3683-0>.
  44. Tian, H., Han, X., Wu, G., Zhou, Y., Li, S., Wei, J., Ye, D., Wang, W., and Zhang, T. (2024). An LLM-enhanced Multi-objective Evolutionary Search for Autonomous Driving Test Scenario Generation. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2406.10857>.
  45. Ye, H., Wang, J., Cao, Z., and Song, G. (2024). ReEvo: Large Language Models as Hyper-Heuristics with Reflective Evolution. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2402.01145>.
  46. Xu, L., Li, A., Zhu, L., Xue, H., Zhu, C., Zhao, K., He, H., Zhang, X., Kang, Q., and Lan, Z. (2023). SuperCLUE: A Comprehensive Chinese Large Language Model Benchmark. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2307.15020>.
  47. UC Irvine (2023). UCI Machine Learning Repository. archive.ics.uci.edu. <https://archive.ics.uci.edu>.
  48. White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J., and Schmidt, D.C. (2023). A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2302.11382>.
  49. Takeuchi, T. (2024). toshikait/MatGPT. GitHub. <https://github.com/toshikait/MatGPT>.
  50. OpenAI (2024). Assistants API. Openai.com. <https://platform.openai.com/docs/assistants/overview>.
  51. Harada, T., and Alba, E. (2021). Parallel Genetic Algorithms: A Useful Survey. *ACM Comput. Surv.* 53, 1–39. <https://doi.org/10.1145/3400031>.
  52. Jun-ichi, K., Hara, A., Takahama, T., and Kido, A. (2013). Island-based differential evolution with varying subpopulation size. In 2013 IEEE 6th International Workshop on Computational Intelligence and Applications (IWCIA) (IEEE), pp. 119–124. <https://doi.org/10.1109/iwcia.2013.6624798>.
  53. Li, C., Yang, S. (2008). An Island Based Hybrid Evolutionary Algorithm for Optimization. Proceedings of the 7th International Conference on Simulated Evolution and Learning. In Springer-Verlag., X. Li, M. Kirley, M. Zhang, D. Green, V. Ciesielski, H. Abbass, Z. Michalewicz, T. Hendtlass, K. Deb, K.C. Tan, J. Branke, Y. Shi, eds., pp. 180–189. [https://doi.org/10.1007/978-3-540-89694-4\\_19](https://doi.org/10.1007/978-3-540-89694-4_19).
  54. Gong, Y., and Fukunaga, A. (2011). Distributed island-model genetic algorithms using heterogeneous parameter settings. In 2011 IEEE Congress of Evolutionary Computation (CEC) (IEEE), pp. 820–827. <https://doi.org/10.1109/cec.2011.5949703>.
  55. Novocontrol Technologies (2023). Alpha-A Analyzer. Novocontrol.de. [https://www.novocontrol.de/php/ana\\_alpha.php](https://www.novocontrol.de/php/ana_alpha.php).
  56. Tektronix (2024). 2600B Series SMU. Tek.com. <https://www.tek.com/en/products/keithley/source-measure-units/2600b-series-sourcemeter>.
  57. National Instruments (2024). GPIB-USB-HS. Ni.com. <https://www.ni.com/en-us/shop/model/gpib-usb-hs.html>.
  58. Zhang, M., Press, O., Merrill, W., Liu, A., and Smith, N.A. (2023). How Language Model Hallucinations Can Snowball. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2305.13534>.
  59. Liu, Y. (2024). RePower source code and experimental data (1.0). Zenodo. <https://doi.org/10.5281/zenodo.14216501>.
  60. Liu, Y. (2024). ChatGPT conversation examples of 6 tasks. Zenodo. <https://doi.org/10.5281/zenodo.14219850>.

**Supplemental information**

**RePower: An LLM-driven autonomous platform  
for power system data-guided research**

**Yu-Xiao Liu, Mengshuo Jia, Yong-Xin Zhang, Jianxiao Wang, Guannan He, Shao-Long  
Zhong, and Zhi-Min Dang**

## A Supplemental Notes

### Note S1 - Preliminary preparations of the system

Unlike task preparation, preliminary preparation of the system here refers to the preparation for constructing and operating the entire system. Most of the code workload of the system is in the application programming interface (API) and the Transport Layer. What kind of API to use depends on the specific application, as for this work, we use MATLAB-python API[S1] to execute the MATLAB code. It is worth noting that the MATLAB environment is needed if the system is deployed locally because of the requirements of the MATLAB-python engine (Figure S4).

The Transport Layer mainly create the client of LLM, exchange data between the client and the output of the MATLAB-python engine, and use the defined API. In this study, all this is achieved in the environment of Jupyter Notebook (Anaconda3)[S2], using python language.

There are mainly two testing platforms and environments, which is a Thinkbook computer AMD Ryzen 7 7735H @3.20GHz (8 cores and 16 threads) attached with Radeon Graphics (2GB) and 16GB RAM, and a working station with intel(R) Xeon(R) Platinum 8372C CPU@3.20GHz (56 cores and 56 threads), 512GB RAM and NVIDIA RTX A6000 (48GB GDDR6) respectively. Considering the lightweight deployment of the system, most tasks can be directly completed in the former, while some tasks that run for too long can be completed in the latter.

### Note S2 - Data collection of the three virtual tasks

Data of all virtual tasks are obtained in advance. The following texts describe the data acquisition process of each virtual task, including the original code or specific experimental operations. We can regard this step as the construction process of **expert knowledge base**, although it is not very appropriate.

#### Task 1: Capacitor capacitance prediction

We used the modified biaxially oriented polypropylene (BOPP)[S3] film independently developed in our laboratory to stretch and roll to prepare capacitors, and then used a digital bridge to measure the capacitance changes of the five capacitors over 70 days at room temperature and indoor humidity[S4]. This task is all about experimental data, therefore the amount of data will inevitably be relatively small.

#### Task 2: Optimization of capacitor site selection and capacity determination

Essentially this problem is an optimization problem in power systems, and it is difficult to solve and obtain the results if we do not use any ready-made solvers. So out of this consideration, particle swarm optimization (PSO) algorithm combined with MATPOWER toolbox are used to calculate the power loss by traversing through all possibilities (noting that some inputs are continuous variables, we only use some specific values for discretization). Below is part of the original codes (incomplete function):

```
while(iterations<10)
    %% define
    particlesize=50; % Number of particles
    Max_num=11; % PSO maximum number of iterations
    c1=1;
    c2=1;
    w=1;
    vcc=2;
    vmax=0.2;
    x_0=zeros(particlesize,6); % Normalized position
    % Decoded position
```

```

x=zeros(particlesize,6); % The six variables correspond to the series compensation point, the size of the series resistance, the parallel compensation point 1, the parallel compensation 1 capacity, the parallel compensation point 2, and the parallel compensation 2 capacity.
v=zeros(particlesize,6); % Speed initialization
f=zeros(particlesize,1); % Fitness function of 3 objectives
g=zeros(particlesize,1); % The particle fitness function that only considers active power loss is not used as a particle update criterion, but this part of the data is needed
%% Initialize particle position
% Decoding
x_0=hundun(particlesize,6);
x=decode(x_0,my_set);
% Initialization speed
v=-0.15+[hundun(particlesize) hundun(particlesize) hundun(particlesize) hundun(particlesize) hundun(particlesize) hundun(particlesize)]*0.30;
for jj=1:particlesize
    [f(jj),g(jj)]=tea2(x(jj,1),x(jj,2),x(jj,3),x(jj,4),x(jj,5),x(jj,6)); % Calculate fitness
end
personalbest_x=x_0; % Use original encoding
personalbest_f=f; % Take it as the individual optimal solution
% Find the overall optimal solution
[groupbest_f0, i]=min(f);
groupbest_x=x_0(i,:);
[groupbest_ploss, i]=min(g); % Find optimal active power loss in particles
j_Num=1;
best_f=zeros(Max_num+1,1);
best_f(j_Num)=groupbest_f0;
for j_Num=2:Max_num+1
    vc=vcc*(Max_num-j_Num)/Max_num+0.2;
    % speed limit
    for kk=1:particlesize
        for kk0=1:6
            if v(kk,kk0)>vmax
                v(kk,kk0)=vmax;
            else
                if v(kk,kk0)<-vmax
                    v(kk,kk0)=-vmax;
                end
            end
        end
    end
    x_0=x_0+vc*v;
    % Boundary restrictions, introducing random positions after exceeding the boundary
    for kk=1:particlesize
        for kk0=1:6
            if ((x_0(kk,kk0)>=1)||((x_0(kk,kk0)<=0))
                x_0(kk,kk0)=rand;
            end
        end
    end
end

```

```

    end
    % Calculate the fitness of each particle
    % Decoding
    x=decode(x_0,my_set);
    for jjj=1:particlesize
        [f(jjj),g(jjj)]=tea2(x(jjj,1),x(jjj,2),x(jjj,3),x(jjj,4),x(jjj,5),x(jjj,6)); % Update particle fitness
    end

    for kk=1:particlesize
        if f(kk)<personalbest_f(kk)
            personalbest_f(kk)=f(kk);
            personalbest_x(kk)=x_0(kk);
        end
    end
    [groupbest_f0 ,i]=min(personalbest_f);
    groupbest_x=x_0(i,:);
    [bb,ii]=sort(f);
    for kk1=1:10
        for kk2=1:6
            x_0(ii(kk1),kk2)=x_0(ii(kk1),kk2)+0.05*x_0(ii(kk1),kk2)*rand();
            if x_0(ii(kk1),kk2)>=1
                x_0(ii(kk1),kk2)=rand();
            end
        end
    end
    best_f(j_Num)=groupbest_f0;
    [groupbest_ploss, i]=min(g); % Find optimal active power loss in particles
end
iterations=iterations+1;
end

```

It is obvious that the above algorithm cannot obtain the global optimal solution, so we will run the code for several times to get a reasonable range of the power loss. All subsequent work will be based on the range rather than a specific number.

### Task 3: Capacitor capacitance prediction

This is a very common and popular problem in the field of power electronics. Generally, there are only two ways to obtain data: experimental testing or simulation. Here we adopt the latter because the speed of obtaining data will be faster and the simulation platform can be quickly built with SIMULINK.

DC-link capacitor is located between bridge rectifier circuit and inverter circuit. In our simulation models, we use the three-phase asynchronous motor as load and a corresponding control circuit is added[S5]. The capacitor here is not an ideal component as we want to consider the aging process, so we add a series resistor to the capacitor. The simulation diagram is illustrated in Figure S5.

### Note S3 - Selection of LLM and tasks

There are many mature large language models on the market, but here we choose GPT-3.5-turbo-0125 as the main tool to fulfill all tasks because GPT-3.5 can be considered as a comparative baseline for existing LLM, which is also the approach used in many papers. It is obvious the fact that this LLM is not the state of art, but the most representative to demonstrate the feasibility of our system. During the later construction process of our system, GPT-4o appeared, so we also attempted to apply it to our system. But its generative

performance on our task did not improve much, both of which make similar mistakes listed in Examples of failure generations, so we ultimately used GPT-3.5.

We have also made relevant attempts in other LLMs besides the GPT family such as Yi-large (which is exactly the same calling method as GPT), and similar results can also be achieved. The comparison results of these LLMs are shown as in Figure S10. Changing the LLM will not affect the framework implementation of the entire system because after the specified number of evolution rounds, the performances of different large-scale models are similar. Instead, it will have an impact on the effect of a single generation, and ultimately only affect the running time of the framework or the number of iterations we set. Besides, in the main text, it is also mentioned that the purpose of using LLM is to iteratively optimize and find better methods. As long as one type of LLM can achieve this effect, the goal is achieved, regardless of the type of LLMs.

The essence of the problems that can be solved by a system like ours is to abstract them into purely data-driven (similar to supervised learning) problems, which basically, has given input and output conditions, and a certain amount of data (labeled samples). As shown in Figure S11, we extracted keywords from the data-driven power system-related literature and selected three representative tasks: optimization corresponding to the power optimization in the main text, fault diagnosis corresponding to state estimation, and prediction model corresponding to parameter prediction. The three tasks we designed in the main text are all related to capacitors to a certain extent, but this is just the tip of the iceberg in this industry or power systems. The reason for our design is twofold: firstly, it is the research direction of our research group; secondly, we want to integrate digital technology into traditional device development research to achieve industry transformation. Although it is very difficult or questioned by many, it is undoubtedly a new quality productivity for scientific research.

We were excited seeing that our system performs well on a benchmark LLM and interested in seeing how the system behaves not with 3 preset tasks, but many more. Therefore, we attempt to use our framework to solve more problems in power systems or other fields to demonstrate its extensive generalization ability. We found three very suitable data-driven tasks in the UCI Machine Learning Repository[S6]-[S8] and obtained their data, attempting to design and evolve solutions using RePower. We named these three tasks Task 4-6.

Task 4 is to transform it into an end-to-end optimization problem similar to task 2 in our main text, using optimization parameters (site and power of 49 wave energy converters) to calculate efficiency indicators and maximum power generation. Task 5 is to use the design of 2D elastodynamic material crystal cells to calculate the bandgap position and width. Task 6 involves calculating the system's stability based on control parameters from four nodes and categorizing whether the system is stable or not, that is, this task can be regarded as a combination of regression and classification. The dimensions of the input and output data for all tasks are as shown in Table S1, while the meanings of the output data have been thoroughly explained in the text.

#### **Task 4: Optimization of site selection for wave energy converters**

◆Example of generations:

→ whole code: [https://zenodo.org/records/14219850/files/task\\_4.png](https://zenodo.org/records/14219850/files/task_4.png)

→ function only: (Classify according to the number of generated models, In the actual iteration process, the latter is too time-consuming, so the former is chosen.)

Single: [https://zenodo.org/records/14219850/files/task\\_4\\_function\\_single\\_model.png](https://zenodo.org/records/14219850/files/task_4_function_single_model.png)

Multi: [https://zenodo.org/records/14219850/files/task\\_4\\_function\\_multi\\_models.png](https://zenodo.org/records/14219850/files/task_4_function_multi_models.png)

#### **Task 5: Metamaterial design and performance prediction**

◆Example of generations:

→ whole code: [https://zenodo.org/records/14219850/files/task\\_5.png](https://zenodo.org/records/14219850/files/task_5.png)

→ function only: [https://zenodo.org/records/14219850/files/task\\_5\\_function.png](https://zenodo.org/records/14219850/files/task_5_function.png)

#### **Task 6: Power system stability control and calculation**

◆Example of generations:

→ whole code: [https://zenodo.org/records/14219850/files/task\\_6.png](https://zenodo.org/records/14219850/files/task_6.png)

→ function only: [https://zenodo.org/records/14219850/files/task\\_6\\_function.png](https://zenodo.org/records/14219850/files/task_6_function.png)

The description of these two tasks and the effect of iterative optimization are shown in Figure S12. Unlike the previous three tasks, Task 4 involves more optimization parameters (98 inputs) and more outputs (50 outputs) while the input for Task 5 is a binary encoded crystal cell structure, and both tasks need to handle large volumes of data which leads to a sharp increase in task difficulty (We have marked the complexity level with a plus icon in the diagram.). Our preliminary tests have revealed that due to the

increased difficulty of the task, relying solely on the description of the task is virtually impossible to generate correct code, let alone subsequent optimization. Therefore, functional frameworks were used to generate these frameworks, which were jointly constructed by humans and LLM as described in the main text. Additionally, since the processing time of these two tasks is relatively long, the hyperparameter search process in the first stage is removed from the entire system, which does not affect the overall search process. Task 6 is relatively simple, yet it encompasses both prediction and classification tasks, which can further test the performance capabilities of the system.

The optimization results of these three additional tasks are also in line with expectations and can achieve continuous optimization. Compared with the previous three tasks, there are relatively few optimizations implemented in the search process in the second stage. And if we look at the magnitude of the RMSE, we can also see that it is significantly larger than the first three tasks. This is because the original data has a higher volatility, with multiple values having large deviations corresponding to the same data. These three databases were not created specifically for prediction, so they may not necessarily achieve good results in prediction. However, the methods we eventually found, the random forest for Task 4, the decision tree ensemble for Task 5, the gaussian process regression and the SVM for Task 6, were all able to perform well in this scenario, outperforming other methods in our searching process by RePower.

It can be discovered from such a system that the potential of large language models (LLMs) is almost infinite. The only drawback is that these tasks are still data-driven and to some extent disconnected from physical objects or laws. However, we believe that with the development of technology, large language models may also be able to complete the task of perception in the physical world, and at that time, they will truly unleash productivity-enhancing effects of artificial intelligence.

## Note S4 - Complete operation process for three virtual tasks

The complete operation result consists of two stages. Due to the excessive number of samples in the second stage, which is not convenient to display, the complete generation mentioned here refers to the first stage.

### Task 1: Capacitor capacitance prediction.

◆Example of generations (May be inconsistent with what is presented later):

→ whole code: [https://zenodo.org/records/14219850/files/task\\_1.png](https://zenodo.org/records/14219850/files/task_1.png)

→ function only: [https://zenodo.org/records/14219850/files/task\\_1\\_function.png](https://zenodo.org/records/14219850/files/task_1_function.png)

**System (Base Knowledge):** you are a data prediction algorithm scientist and optimizer. When the code needs to be modified, it is necessary to provide the complete code, including the part without errors. In the code there must exist an auto parameter optimizing process which means the algorithm will automatically find the best parameter to minimize the prediction error.

### **Task (Admin):**

Version 1  Whole code generations	<p>I have data on the capacitance of capacitors over 70 days. I hope to learn the capacitance data from the first 60 days, and then predict the capacitance data for the remaining 10 days. In addition, the prediction effect is tested through average RMSE in the remaining 10 days. To solve this problem, please choose the proper method, and write the corresponding MATLAB program. Add the full capacitance data in the code according to the data in the prompt.</p> <p>The specific data on the variation of capacitance value with days is as follows:</p> <pre>C1I = [27.547, 27.52, 27.508, 27.488, 27.516, 27.518, 27.471, 27.435, 27.437, 27.431, 27.455, 27.46, 27.373, 27.352, 27.336, 27.297, 27.295, 27.285, 27.273, 27.278, 27.271, 27.287, 27.33, 27.342, 27.374, 27.356, 27.345, 27.345, 27.336, 27.333, 27.341, 27.38, 27.369, 27.343, 27.314, 27.274, 27.275, 27.256, 27.257, 27.252, 27.247, 27.23, 27.222, 27.218, 27.24, 27.243, 27.244, 27.214, 27.2, 27.172, 27.174, 27.156, 27.169, 27.18, 27.169, 27.167, 27.193, 27.141, 27.176, 27.153, 27.161, 27.146, 27.14, 27.124, 27.142, 27.142, 27.136, 27.144, 27.152, 27.11];</pre> <pre>C2I = [27.683, 27.657, 27.646, 27.626, 27.657, 27.657, 27.609, 27.576, 27.581,</pre>
---	--

```

27.574, 27.599, 27.602, 27.517, 27.501, 27.485, 27.446, 27.443, 27.433, 27.422,
27.428, 27.419, 27.434, 27.474, 27.486, 27.517, 27.501, 27.49, 27.49, 27.483,
27.48, 27.489, 27.523, 27.514, 27.488, 27.458, 27.425, 27.424, 27.407, 27.407,
27.403, 27.397, 27.383, 27.373, 27.37, 27.39, 27.393, 27.394, 27.363, 27.348,
27.322, 27.325, 27.308, 27.318, 27.333, 27.321, 27.318, 27.343, 27.294, 27.328,
27.303, 27.313, 27.298, 27.293, 27.276, 27.294, 27.295, 27.288, 27.296, 27.302,
27.26];
C3I = [27.593, 27.565, 27.554, 27.535, 27.565, 27.564, 27.516, 27.482, 27.485,
27.479, 27.503, 27.507, 27.42, 27.402, 27.385, 27.346, 27.343, 27.332, 27.322,
27.327, 27.319, 27.335, 27.374, 27.386, 27.418, 27.401, 27.394, 27.391, 27.384,
27.383, 27.393, 27.426, 27.418, 27.393, 27.362, 27.326, 27.325, 27.306, 27.305,
27.303, 27.295, 27.281, 27.272, 27.27, 27.29, 27.295, 27.295, 27.265, 27.248,
27.222, 27.224, 27.207, 27.221, 27.234, 27.221, 27.219, 27.245, 27.194, 27.227,
27.203, 27.213, 27.199, 27.193, 27.176, 27.194, 27.195, 27.188, 27.187, 27.202,
27.158];
C4I = [27.58, 27.552, 27.543, 27.522, 27.553, 27.552, 27.506, 27.471, 27.475,
27.47, 27.493, 27.498, 27.412, 27.393, 27.378, 27.342, 27.338, 27.327, 27.318,
27.322, 27.314, 27.331, 27.368, 27.378, 27.409, 27.394, 27.386, 27.381, 27.376,
27.375, 27.386, 27.415, 27.407, 27.385, 27.355, 27.322, 27.317, 27.3, 27.298,
27.294, 27.288, 27.274, 27.265, 27.265, 27.283, 27.288, 27.287, 27.259, 27.244,
27.217, 27.218, 27.203, 27.215, 27.23, 27.216, 27.214, 27.238, 27.19, 27.221,
27.197, 27.206, 27.194, 27.189, 27.171, 27.188, 27.188, 27.182, 27.194, 27.198,
27.155];
C5I = [27.661, 27.633, 27.625, 27.604, 27.635, 27.635, 27.587, 27.555, 27.559,
27.553, 27.577, 27.581, 27.5, 27.481, 27.465, 27.429, 27.425, 27.414, 27.404,
27.409, 27.401, 27.418, 27.452, 27.462, 27.494, 27.479, 27.47, 27.467, 27.461,
27.46, 27.471, 27.5, 27.492, 27.471, 27.439, 27.408, 27.404, 27.388, 27.387,
27.382, 27.376, 27.363, 27.353, 27.353, 27.37, 27.375, 27.374, 27.347, 27.332,
27.304, 27.305, 27.292, 27.304, 27.32, 27.304, 27.302, 27.325, 27.279, 27.31,
27.288, 27.296, 27.283, 27.279, 27.262, 27.277, 27.277, 27.273, 27.281, 27.288,
27.244];

```

Can the current algorithm be further optimized or you can try another method?

<p>Version 2</p> <p>Prediction function only</p>	<p>I have data on the capacitance of capacitors over 70 days. I hope to learn the capacitance data from the first 60 days, and then predict the capacitance data for the remaining 10 days. In addition, the prediction effect is tested through average RMSE in the remaining 10 days. To solve this problem, please choose the proper method, and complete the corresponding MATLAB prediction function.</p> <p>The specific codes are as follows:</p> <pre>```matlab C1I = [27.547, 27.52, 27.508, 27.488, 27.516, 27.518, 27.471, 27.435, 27.437, 27.431, 27.455, 27.46, 27.373, 27.352, 27.336, 27.297, 27.295, 27.285, 27.273, 27.278, 27.271, 27.287, 27.33, 27.342, 27.374, 27.356, 27.345, 27.345, 27.336, 27.333, 27.341, 27.38, 27.369, 27.343, 27.314, 27.274, 27.275, 27.256, 27.257, 27.252, 27.247, 27.23, 27.222, 27.218, 27.24, 27.243, 27.244, 27.214, 27.2, 27.172, 27.174, 27.156, 27.169, 27.18, 27.169, 27.167, 27.193, 27.141, 27.176, 27.153, 27.161, 27.146, 27.14, 27.124, 27.142, 27.142, 27.136, 27.144, 27.152, 27.11]; C2I = [27.683, 27.657, 27.646, 27.626, 27.657, 27.657, 27.609, 27.576, 27.581, 27.574, 27.599, 27.602, 27.517, 27.501, 27.485, 27.446, 27.443, 27.433, 27.422, 27.428, 27.419, 27.434, 27.474, 27.486, 27.517, 27.501, 27.49, 27.49, 27.483, 27.48, 27.489, 27.523, 27.514, 27.488, 27.458, 27.425, 27.424, 27.407, 27.407, 27.403, 27.397, 27.383, 27.373, 27.37, 27.39, 27.393, 27.394, 27.363, 27.348, 27.322, 27.325, 27.308, 27.318, 27.333, 27.321, 27.318, 27.343, 27.294, 27.328, 27.303, 27.313, 27.298, 27.293, 27.276, 27.294, 27.295, 27.288, 27.296, 27.302, 27.26]; C3I = [27.593, 27.565, 27.554, 27.535, 27.565, 27.564, 27.516, 27.482, 27.485, 27.479, 27.503, 27.507, 27.42, 27.402, 27.385, 27.346, 27.343, 27.332, 27.322, 27.327, 27.319, 27.335, 27.374, 27.386, 27.418, 27.401, 27.394, 27.391, 27.384, 27.383, 27.393, 27.426, 27.418, 27.393, 27.362, 27.326, 27.325, 27.306, 27.305, 27.303, 27.295, 27.281, 27.272, 27.27, 27.29, 27.295, 27.295, 27.265, 27.248, 27.222, 27.224, 27.207, 27.221, 27.234, 27.221, 27.219, 27.245, 27.194, 27.227, 27.203, 27.213, 27.199, 27.193, 27.176, 27.194, 27.195, 27.188, 27.187, 27.202, 27.158]; C4I = [27.58, 27.552, 27.543, 27.522, 27.553, 27.552, 27.506, 27.471, 27.475, 27.47, 27.493, 27.498, 27.412, 27.393, 27.378, 27.342, 27.338, 27.327, 27.318, 27.322, 27.314, 27.331, 27.368, 27.378, 27.409, 27.394, 27.386, 27.381, 27.376, 27.375, 27.386, 27.415, 27.407, 27.385, 27.355, 27.322, 27.317, 27.3, 27.298, 27.294, 27.288, 27.274, 27.265, 27.265, 27.283, 27.288, 27.287, 27.259, 27.244, 27.217, 27.218, 27.203, 27.215, 27.23, 27.216, 27.214, 27.238, 27.19, 27.221, 27.197, 27.206, 27.194, 27.189, 27.171, 27.188, 27.188, 27.182, 27.194, 27.198, 27.155]; C5I = [27.661, 27.633, 27.625, 27.604, 27.635, 27.635, 27.587, 27.555, 27.559, 27.553, 27.577, 27.581, 27.5, 27.481, 27.465, 27.429, 27.425, 27.414, 27.404, 27.409, 27.401, 27.418, 27.452, 27.462, 27.494, 27.479, 27.47, 27.467, 27.461, 27.46, 27.471, 27.5, 27.492, 27.471, 27.439, 27.408, 27.404, 27.388, 27.387, 27.382, 27.376, 27.363, 27.353, 27.353, 27.37, 27.375, 27.374, 27.347, 27.332, 27.304, 27.305, 27.292, 27.304, 27.32, 27.304, 27.302, 27.325, 27.279, 27.31, 27.288, 27.296, 27.283, 27.279, 27.262, 27.277, 27.277, 27.273, 27.281, 27.288, 27.244]; ``` </pre>
--	---

```

% Combine data into a matrix
data = [C1I; C2I; C3I; C4I; C5I]';

% Prepare the data for training and testing
x_train = (1:60)';
y_train = data(1:60,:);
x_test = (61:70)';
y_test = data(61:70,:);

avgRMSE = prediction(x_train,y_train,x_test,y_test);

% Display the result
fprintf('Average RMSE over the 10 days: %.4f\n', avgRMSE);

% Function to calculate RMSE
function rmse = calculateRMSE(predicted, actual)
    rmse = sqrt(mean((predicted - actual).^2));
end

% Function to predict the results
function avgRMSE = prediction(x_train,y_train,x_test,y_test)

    % complete the blank

end
...
please remember that you should generate the prediction function only and do not redefine the 'function rmse = calculateRMSE(predicted, actual)'.

```

Try to use another prediction method and add the automated parameter searching step.

**Response:** To predict the capacitance data for the remaining 10 days, I recommend using linear regression. Here is the completed MATLAB prediction function using linear regression:

```

```matlab
function avgRMSE = prediction(x_train,y_train,x_test,y_test)

num_samples = size(y_train, 2);
predictions = zeros(numel(x_test), num_samples);

for i = 1:num_samples
    % Fit linear regression model
    mdl = fitlm(x_train, y_train(:,i));

    % Predict capacitance for the test period
    y_pred = predict(mdl, x_test);

```

```

predictions(:,i) = y_pred;
end

% Calculate RMSE for each sample
RMSE_values = zeros(num_samples, 1);
for i = 1:num_samples
RMSE_values(i) = calculateRMSE(predictions(:,i), y_test(:,i));
end

% Calculate average RMSE over all samples
avgRMSE = mean(RMSE_values);
end
...

```

In this function, we iterate over each sample (each capacitor) in the dataset, fit a linear regression model using the training data, predict the capacitance values for the test period, calculate RMSE for each sample, and finally, compute the average RMSE over all samples. This approach allows us to predict the capacitance data for the remaining 10 days using the first 60 days as training data.

**MATLAB Assistant:** mes = no error

res\_1 = Average RMSE over the 10 days: 0.032543

(MATLAB Assistant will be omitted in the following text because in most cases, their information overlaps with the user)

**User:** The current RMSE is 0.032543. I want to make the RMSE lower than 0.01. Try to use another prediction method and add the automated parameter searching step.

**Response:** I apologize for the oversight. Let's update the prediction function using SVM regression without specifying the KernelFunction to address the error. In this case, MATLAB will internally select the appropriate kernel function based on the data.

Here is the revised MATLAB prediction function:

```

```matlab
function avgRMSE = prediction(x_train,y_train,x_test,y_test)

num_samples = size(y_train, 2);
predictions = zeros(numel(x_test), num_samples);

for i = 1:num_samples
    % Fit SVM regression model
    mdl = fitrsvm(x_train, y_train(:,i));

    % Predict capacitance for the test period
    y_pred = predict(mdl, x_test);
    predictions(:,i) = y_pred;
end

% Calculate RMSE for each sample
RMSE_values = zeros(num_samples, 1);
for i = 1:num_samples
RMSE_values(i) = calculateRMSE(predictions(:,i), y_test(:,i));

```

```

end

% Calculate average RMSE over all samples
avgRMSE = mean(RMSE_values);
end
```

```

With this update, MATLAB will internally select the appropriate kernel function for the SVM regression model, which should help to resolve the error related to the 'KernelFunction' parameter. Let me know if you encounter any further issues.

**User:** The current RMSE is 0.037547. I want to make the RMSE lower than 0.01. Try to use another prediction method and add the automated parameter searching step.

**Response:** I see the issue now. Let's simplify the function and ensure that the number of arguments matches the function definition. Here is the updated prediction function:

```

```matlab
function avgRMSE = prediction(x_train, y_train, x_test, y_test)

num_samples = size(y_train, 2);
predictions = zeros(numel(x_test), num_samples);

for i = 1:num_samples
    rng(1); % For reproducibility

    % Train the Gradient Boosting Regressor model
    mdl = fitrensemble(x_train, y_train(:,i), 'Method', 'LSBoost', 'NumLearningCycles', 150, 'LearnRate', 0.1);

    % Predict the capacitance for the test period
    y_pred = predict(mdl, x_test);
    predictions(:,i) = y_pred;
end

% Calculate RMSE for each sample
RMSE_values = zeros(num_samples, 1);
for i = 1:num_samples
    RMSE_values(i) = calculateRMSE(predictions(:,i), y_test(:,i));
end

% Calculate average RMSE over all samples
avgRMSE = mean(RMSE_values);
end
```

```

In this version, the function takes four arguments: `x_train`, `y_train`, `x_test`, and `y_test`, corresponding to the training data, testing data, and input samples for the prediction. The function then trains a Gradient Boosting Regressor model for each sample, makes predictions for the test period, calculates the RMSE for each sample, and computes the average RMSE over all samples.

Please try running this revised prediction function with the provided arguments, and let me know if you encounter any issues.

**User:** The current RMSE is 0.029827. I want to make the RMSE lower than 0.01. Try to use another

prediction method and add the automated parameter searching step.

...

### Task 2: Optimization of capacitor site selection and capacity determination.

◆ Example of generations (May be inconsistent with what is presented later):

→ whole code: [https://zenodo.org/records/14219850/files/task\\_2.png](https://zenodo.org/records/14219850/files/task_2.png)

→ function only: [https://zenodo.org/records/14219850/files/task\\_2\\_function.png](https://zenodo.org/records/14219850/files/task_2_function.png)

**System (Base Knowledge):** you are a data prediction algorithm scientist and optimizer. When the code needs to be modified, it is necessary to provide the complete code, including the part without errors. In the code there must exist an auto parameter optimizing process which means the algorithm will automatically find the best parameter to minimize the prediction error.

#### **Task (Admin):**

|                                                                                 |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                            |
|---------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Version 1<br><br>Whole code                                                     | You are a data analyst and predictor, and need to establish the correlation between the first 9 columns of data and the last two columns of data based on the data I provide. The first 9 columns are optimization parameters and the last two columns are Ploss max and Ploss min respectively. When conducting model training, it is necessary to provide the training and testing sets. In addition, the prediction effect is tested through average RMSE in Ploss max and Ploss min. To solve this problem, please choose the proper method, and write the corresponding MATLAB program. The required data files are saved in all xlsx files in the folder named data_1 in the upper-level directory'..data_1', the first line of the file is the column name, and the second line starts with the data.<br>Do not use fitlm function.                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                 |
| Version 2<br><br>clear instruction with background information<br>function only | You are a data analyst and predictor. There is now a dataset stored in all xlsx files in the upper-level directory'..data_1', which is obtained through particle swarm optimization and optimal power flow calculation in IEEE33 node for the reactive power compensation problem of capacitors. The data in the file is divided into 11 columns, with the first 9 columns (representing series compensation position, series compensation length, series compensation capacity, parallel compensation position 1, parallel compensation length 1, parallel compensation capacity 1, parallel compensation position 2, parallel compensation length 2, parallel compensation capacity 2) being optimization parameters and the last 2 columns being the calculated maximum and minimum values of active power. Your goal is to build end-to-end mapping functions for the first 9 columns of parameters and the last 2 columns of parameters<br>You need to divide the data in this dataset into training and testing sets. In addition, the predicted results need to be presented through the RMSE of the maximum and minimum active power values, which is calculated as the average of the predicted results in the testing set. To solve this problem, please choose the proper method, and complete the corresponding MATLAB prediction function.<br>The specific codes are as follows:<br><pre>```matlab % Load data from xlsx files in the 'data_1' directory fileList = dir('..data_1/* .xlsx'); data = [];</pre> |

```

for i = 1:length(fileList)
    filename = fullfile(fileList(i).folder, fileList(i).name);
    newData = readmatrix(filename);
    data = [data; newData];
end

% Split the data into input (parameters) and output (active power values)
variables
x = data(:, 1:9);
y = data(:, 10:11);

% Divide the data into training and testing sets
rng(42); % for reproducibility
cv = cvpartition(size(data,1),'HoldOut',0.3);
idx = cv.test;

x_train = x(~idx, :);
y_train = y(~idx, :);
x_test = x(idx, :);
y_test = y(idx, :);

[avgRMSE_max,avgRMSE_min] = prediction(x_train,y_train,x_test,y_test);

% Display the result
fprintf('Average RMSE for maximum active power prediction: %.4f',
avgRMSE_max);
fprintf('Average RMSE for minimum active power prediction: %.4f',
avgRMSE_min);

% Function to calculate RMSE
function rmse = calculateRMSE(predicted, actual)
    rmse = sqrt(mean((predicted - actual).^2));
end

% Function to predict the results
function [avgRMSE_max,avgRMSE_min] =
prediction(x_train,y_train,x_test,y_test)

    % complete the blank

    % Calculate average RMSE
    avgRMSE_max = mean(RMSE_max);
    avgRMSE_min = mean(RMSE_min);
end
```
please remember that you should generate the prediction function only and do not redefine the 'function rmse = calculateRMSE(predicted, actual)'.

```

**Response:**

```
```matlab
% Function to predict the results
function [avgRMSE_max, avgRMSE_min] = prediction(x_train, y_train, x_test, y_test)

    % Train a regression model - Support Vector Machine (SVM) for example
    mdl = fitrsvm(x_train, y_train(:,1)); % Fit a model for maximum active power prediction
    y_pred_max = predict(mdl, x_test); % Predict maximum active power values

    mdl = fitrsvm(x_train, y_train(:,2)); % Fit a model for minimum active power prediction
    y_pred_min = predict(mdl, x_test); % Predict minimum active power values

    % Calculate RMSE for maximum and minimum active power predictions
    RMSE_max = sqrt(mean((y_pred_max - y_test(:,1)).^2));
    RMSE_min = sqrt(mean((y_pred_min - y_test(:,2)).^2));

    % Calculate average RMSE
    avgRMSE_max = mean(RMSE_max);
    avgRMSE_min = mean(RMSE_min);
end
...```

```

In this updated "prediction" function, Support Vector Machine (SVM) regression models are trained for predicting the maximum and minimum active power values based on the input parameters. The function then calculates the RMSE for both predictions and averages the RMSE values to obtain the final average RMSE for maximum and minimum active power prediction.

**User:** The key step output of the method: Average RMSE for maximum active power prediction: 0.0089092  
Average RMSE for minimum active power prediction: 0.018285

The current standardized RMSE is 0.0135971. I want to make the RMSE lower. Try to use another prediction method and add the automated parameter searching step.

**Response:** I will simplify the prediction function without using Bayesian optimization and implement Random Forest regression directly. Here is the updated prediction function:

```
```matlab
% Function to predict the results with Random Forest regression
function [avgRMSE_max, avgRMSE_min] = prediction(x_train, y_train, x_test, y_test)

    % Train a Random Forest regression model for maximum active power values
    mdl_max = fitrensemble(x_train, y_train(:,1));
    y_pred_max = predict(mdl_max, x_test);

    % Train a Random Forest regression model for minimum active power values
    mdl_min = fitrensemble(x_train, y_train(:,2));
    y_pred_min = predict(mdl_min, x_test);

    % Calculate RMSE for maximum and minimum active power predictions
    RMSE_max = sqrt(mean((y_pred_max - y_test(:,1)).^2));
    RMSE_min = sqrt(mean((y_pred_min - y_test(:,2)).^2));
```

```

```
% Calculate average RMSE
avgRMSE_max = mean(RMSE_max);
avgRMSE_min = mean(RMSE_min);
end
...
```

In this updated prediction function, we are using Random Forest regression models directly without Bayesian optimization. This modification should resolve the "not enough input arguments" error for the provided code.

**User:** The key step output of the method: Average RMSE for maximum active power prediction: 0.0018875

Average RMSE for minimum active power prediction: 0.0010409

The current standardized RMSE is 0.0014642000000000001. I want to make the RMSE lower. Try to use another prediction method and add the automated parameter searching step.

**Response:** I see the issue. Let's revise the prediction function code and make sure it works with the given input arguments:

```
'''matlab
% Function to predict the results with Gradient Boosting regression using Bayesian Optimization
function [avgRMSE_max, avgRMSE_min] = prediction(x_train, y_train, x_test, y_test)

    % Train a Gradient Boosting regression model for maximum active power values
    mdl_max = fitrensemble(x_train, y_train(:,1), 'OptimizeHyperparameters', 'all');
    y_pred_max = predict(mdl_max, x_test);

    % Train a Gradient Boosting regression model for minimum active power values
    mdl_min = fitrensemble(x_train, y_train(:,2), 'OptimizeHyperparameters', 'all');
    y_pred_min = predict(mdl_min, x_test);

    % Calculate RMSE for maximum and minimum active power predictions
    RMSE_max = sqrt(mean((y_pred_max - y_test(:,1)).^2));
    RMSE_min = sqrt(mean((y_pred_min - y_test(:,2)).^2));

    % Calculate average RMSE for maximum and minimum active power predictions
    avgRMSE_max = mean(RMSE_max);
    avgRMSE_min = mean(RMSE_min);
end
'''
```

In this updated prediction function code, I have simplified the process by using default options for the Gradient Boosting regression models. This should ensure that the function can be called with the provided input arguments and successfully perform the predictions for maximum and minimum active power values.

**User:** The key step output of the method: Average RMSE for maximum active power prediction: 0.0027395

Average RMSE for minimum active power prediction: 0.0023399

The current standardized RMSE is 0.0025396999999999998. I want to make the RMSE lower. Try to use another prediction method and add the automated parameter searching step.

...

Task 3: Condition monitoring of the health status of DC-Link capacitors.

◆ Example of generations (May be inconsistent with what is presented later):

→ whole code: [https://zenodo.org/records/14219850/files/task\\_3.png](https://zenodo.org/records/14219850/files/task_3.png)

→ function only: [https://zenodo.org/records/14219850/files/task\\_3\\_function.png](https://zenodo.org/records/14219850/files/task_3_function.png)

**System (Base Knowledge):** you are a MATLAB code generator, and when the code needs to be modified, it is necessary to provide the complete code, including the part without errors.

**Task (Admin):**

|                             |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                    |
|-----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Version 1<br><br>Whole code | You are a data analyst and predictor, and need to establish the correlation between the capacitance (C) and equivalent series resistance (ESR) and the voltage and current sequence based on the data I provide. The goal is to predict C and ESR using the given voltage and current sequence. When conducting model training, it is necessary to provide the training and testing sets. In addition, the prediction effect is tested through average RMSE in C and ESR. To solve this problem, please choose the proper method, and write the corresponding MATLAB program. The required data files are saved in all xlsx files in the folder named data_2 in the upper-level directory'..//data_2/'. The first line of the file is the column name, and the second line starts with the data. The first two columns are voltage and current sequence, the last two columns are the corresponding C and ESR.<br>In each file, the voltage and current data is stored in string matrix form, so first you need to read from the original file by reading cell array and convert the cell into the char matrix using "cell2mat".<br>Voltage and current are 101 dimensional column vectors, and C and ESR are 1-dimensional data, so you need to turn the voltage and current data into row vectors<br>Here is an example to finish the voltage and current data reading and processing<br><pre>voltage_cell = cellfun(@(x)str2num(x), data(2:end, 1), 'UniformOutput', false); current_cell = cellfun(@(x)str2num(x), data(2:end, 2), 'UniformOutput', false); % Convert double cell to numeric vectors voltage_numeric = []; current_numeric = []; for j = 1:size(voltage_cell, 1)     voltage_numeric = [voltage_numeric, voltage_cell{j, :}];     current_numeric = [current_numeric, current_cell{j, :}]; end % Concatenate voltage and current column vectors voltage_data = voltage_numeric'; current_data = current_numeric';</pre> <p>remember to use '' to concat voltage and current column vectors</p> |
|-----------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

|                                       |                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                          |
|---------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <p>Version 2</p> <p>Function only</p> | <p>You are a data analyst and predictor, and need to establish the correlation between the capacitance (C) and equivalent series resistance (ESR) and the voltage and current sequence based on the data I provide. The goal is to predict C and ESR using the given voltage and current sequence. When conducting model training, it is necessary to provide the training and testing sets. In addition, the prediction effect is tested through average RMSE in C and ESR. The required data files are saved in all .xlsx files in the folder named <code>data_2</code> in the upper-level directory <code>'..data_2'</code>. The first line of the file is the column name, and the second line starts with the data. The first two columns are voltage and current sequence, the last two columns are the corresponding C and ESR. To solve this problem, please choose the proper method, and complete the corresponding MATLAB prediction function. The predicted results need to be presented through the RMSE of the C and ESR values, which is calculated as the average of the predicted results in the testing set.</p> <p>The specific codes are as follows:</p> <pre> ```matlab % Specify the folder containing data files data_folder = '..data_2';  % Get list of Excel files in the folder file_list = dir(fullfile(data_folder, '*.xlsx'));  % Initialize arrays to store data voltage_data_all = []; current_data_all = []; C_data_all = []; ESR_data_all = [];  % Loop through each Excel file for i = 1:length(file_list)     file_path = fullfile(data_folder, file_list(i).name);      % Read data from Excel file     data = readcell(file_path);      % Convert voltage and current data from char cell to double cell     voltage_cell = cellfun(@(x)str2num(x), data(2:end, 1), 'UniformOutput', false);     current_cell = cellfun(@(x)str2num(x), data(2:end, 2), 'UniformOutput', false);      % Convert double cell to numeric vectors     voltage_numeric = [];     current_numeric = [];     for j = 1:size(voltage_cell, 1)         voltage_numeric = [voltage_numeric, voltage_cell{j, :}];         current_numeric = [current_numeric, current_cell{j, :}];     end      % Concatenate voltage and current column vectors     voltage_data = voltage_numeric'; </pre> |
|---------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|

```

current_data = current_numeric';

% Read capacitance (C) and equivalent series resistance (ESR) data
C_data = cell2mat(data(2:end, end-1));
ESR_data = cell2mat(data(2:end, end));

% Store data from current file
voltage_data_all = [voltage_data_all; voltage_data];
current_data_all = [current_data_all; current_data];
C_data_all = [C_data_all; C_data];
ESR_data_all = [ESR_data_all; ESR_data];
end

% Combine voltage and current data into one matrix
X = [voltage_data_all, current_data_all];

% Split data into training and testing sets (80% training, 20% testing)
rng(42); % For reproducibility
idx = randperm(size(X, 1));
train_size = round(0.8 * length(idx));
train_idx = idx(1:train_size);
test_idx = idx(train_size+1:end);

X_train = X(train_idx, :);
C_train = C_data_all(train_idx);
ESR_train = ESR_data_all(train_idx);

X_test = X(test_idx, :);
C_test = C_data_all(test_idx);
ESR_test = ESR_data_all(test_idx);

[avgC_rmse,avgESR_rmse] =
prediction(X_train,C_train,ESR_train,X_test,C_test,ESR_test);

% Display the result
fprintf('Average RMSE for capacitance (C): %.4f, avgC_rmse);
fprintf('Average RMSE for ESR: %.4f, avgESR_rmse);

% Function to calculate RMSE
function rmse = calculateRMSE(predicted, actual)
    rmse = sqrt(mean((predicted - actual).^2));
end

% Function to predict the results
function [avgC_rmse,avgESR_rmse] =
prediction(X_train,C_train,ESR_train,X_test,C_test,ESR_test)

```

```
% complete the blank

% Calculate average RMSE
avgC_rmse = mean(C_rmse);
avgESR_rmse = mean(ESR_rmse);
end
...
please remember that you should generate the prediction function only and do not redefine the 'function rmse = calculateRMSE(predicted, actual)'.
```

**Response:**

```
```matlab
% Function to predict the results
function [avgC_rmse,avgESR_rmse] = prediction(X_train,C_train,ESR_train,X_test,C_test,ESR_test)

    % Train a linear regression model to predict capacitance (C) and calculate RMSE
    mdl_C = fitlm(X_train, C_train);
    C_predicted = predict(mdl_C, X_test);
    C_rmse = calculateRMSE(C_predicted, C_test);

    % Train a linear regression model to predict equivalent series resistance (ESR) and calculate RMSE
    mdl_ESR = fitlm(X_train, ESR_train);
    ESR_predicted = predict(mdl_ESR, X_test);
    ESR_rmse = calculateRMSE(ESR_predicted, ESR_test);

    % Calculate average RMSE
    avgC_rmse = mean(C_rmse);
    avgESR_rmse = mean(ESR_rmse);
end
...```

```

In this function, I have completed the code to train linear regression models to predict capacitance (C) and equivalent series resistance (ESR) based on the provided training data. The models are then used to predict C and ESR values for the testing set, and the root mean square error (RMSE) is calculated for each prediction. Finally, the average RMSE for C and ESR values in the testing set is calculated and returned from the function.

**User:** The key step output of the method: Average RMSE for capacitance (C): 1.9784e-05

Average RMSE for ESR: 0.0056305

The current standardized RMSE is 0.01270725. I want to make the RMSE lower. Try to use another prediction method and add the automated parameter searching step.

**Response:** I apologize for the oversight. Let's redefine the prediction function without using the `hyperparameterOptimization` function and utilizing a standard SVM regression model with default hyperparameters. Here is the updated prediction function:

```
```matlab
...```

```

```
% Function to predict the results using SVM regression
function [avgC_rmse, avgESR_rmse] = prediction(X_train, C_train, ESR_train, X_test, C_test, ESR_test)

    % Train SVM regression models
    mdl_C = fitrsvm(X_train, C_train, 'Standardize', true);
    mdl_ESR = fitrsvm(X_train, ESR_train, 'Standardize', true);

    % Predict capacitance (C) and equivalent series resistance (ESR) using SVM models
    C_predicted = predict(mdl_C, X_test);
    ESR_predicted = predict(mdl_ESR, X_test);

    % Calculate RMSE for C and ESR predictions
    C_rmse = calculateRMSE(C_predicted, C_test);
    ESR_rmse = calculateRMSE(ESR_predicted, ESR_test);

    % Calculate average RMSE
    avgC_rmse = mean(C_rmse);
    avgESR_rmse = mean(ESR_rmse);
end
```

```

In this updated version of the prediction function, we are using the default hyperparameters for the SVM regression models. The `fitrsvm` function is used to train the SVM models for predicting capacitance (C) and equivalent series resistance (ESR) values. The RMSE is calculated for both predictions, and the average RMSE values are computed as the output of the function.

Please replace the existing prediction function in your code with this updated version to continue the prediction and evaluation process. Let me know if you need further assistance!

**User:** The key step output of the method: Average RMSE for capacitance (C): 2.5626e-05

Average RMSE for ESR: 0.0073138

The current standardized RMSE is 0.0164699. I want to make the RMSE lower. Try to use another prediction method and add the automated parameter searching step.

**Response:** To address the issue where the output of the prediction function is NaN, we need to handle the cases where the predicted values may contain non-numeric values or invalid predictions. Here is an updated version of the prediction function that includes error handling for non-numeric predictions:

```
'''matlab
function [avgC_rmse, avgESR_rmse] = prediction(X_train, C_train, ESR_train, X_test, C_test, ESR_test)

    % Train Random Forest regression models
    mdl_C = TreeBagger(50, X_train, C_train, 'Method', 'regression');
    mdl_ESR = TreeBagger(50, X_train, ESR_train, 'Method', 'regression');

    % Predict capacitance (C) and equivalent series resistance (ESR) using Random Forest models
    C_predicted = str2double(predict(mdl_C, X_test));
    ESR_predicted = str2double(predict(mdl_ESR, X_test));

    % Handle non-numeric or NaN predictions
end
```

```

```

C_predicted(isnan(C_predicted)) = 0;
ESR_predicted(isnan(ESR_predicted)) = 0;

% Calculate RMSE for C and ESR predictions
C_rmse = calculateRMSE(C_predicted, C_test);
ESR_rmse = calculateRMSE(ESR_predicted, ESR_test);

% Calculate average RMSE
avgC_rmse = mean(C_rmse);
avgESR_rmse = mean(ESR_rmse);
end
...

```

**User:** In this updated version, we convert the predicted values to double using `str2double` to ensure they are numeric. We then handle any NaN values in the predictions by replacing them with 0 before calculating the RMSE. This precaution should help mitigate the issue of getting NaN output for the RMSE calculation.

Please replace the existing prediction function in your code with this updated version and run the analysis again. Let me know if this resolves the NaN output issue or if you need further assistance!

...

## Note S5 - Construction cost

There is a detailed description of the system architecture of the task in the main text, but the requirements or expenses during its construction are not mentioned. Here, a brief introduction will be provided to show the basic cost of building the entire system. The information is depicted in Figure S6.

i. LLM-based system: We have spent more than 50 hours building the initial system, not including subsequent debugging and modification processes. While the building process, approximate 500 attempts are made to ensure the proper operation of each part in our system.

ii. Task 1: Since task 1 was used as a base case for testing when building the system, relatively little instruction time was spent in this task. The generation time is about 20 min, which can be accepted compared to human completion time. The pieces of data is relatively small but the training time is very fast.

ii. Task 2: Similar to task 1, except that the data is read from the outside instead of input directly, so it just takes a little more time to finish the whole work. It can be noted that there is huge amount of data because we adopted a traversal method to get the final optimal value. And the train time, out of the data, becomes very long.

iv. Task 3: The data structure becomes complex, and it takes a long time to achieve the correct conversion of data types. But fortunately, the final training time is relatively short, so the later work efficiency can make up for the early waste.

v. Physical operation: The hardware construction cost is about 10 hours if excluding the equipment purchase cost and time. This takes a long time mainly because we need to learn the operation and control methods of specific equipment. A detailed introduction will be given in the next section.

## Note S6 - Physical operation: combination of hardware and software

**Note:** *This study was performed under strict human supervision. There is a certain security risk in applying the LLM into the physical object, so it is not recommended to hand over all control rights to the LLM especially when the technology is not mature enough.*

In the physical operation part, we only demonstrate controlling three different electronic measuring devices based on our system. And to the safety concern, only a few interface functions are fully open to the LLM while the rest are prohibited.

The following is a brief record of the data acquisition process of the three devices. The blue box represents an example of a task prompt we provide, and it can also be modified as needed.

## Device 1: Impedance analyzer

The impedance analyzer connects directly to the computer via USB A-B, at which point the computer recognizes it as a network port device. By using the corresponding port number, as shown in the connection set in Figure S7A, interaction and control with it can be achieved.

### System (Base Knowledge):

Version 1  Only to conduct normal impedance test	I need you to control the impedance analyzer, first you need to give me two parameters for me to conduct which are time interval for data collection and number of data points respectively. Two parameters are given in the form of the following keywords: TIME_INTERVAL [specific number] s DATA_NUMBER [specific number] p  I want to test for a middle length of time and sampling time should not be too frequent, please give me these two parameters.
Version 2  Add the sweep mode besides the normal test	I need you to control the impedance analyzer, there are basically two models for the user to choose: normal and sweep. If I choose to operate the normal test, you need to give me two parameters for me to conduct which are time interval for data collection and number of data points respectively. Two parameters are given in the form of the following keywords: TIME_INTERVAL [specific number] s DATA_NUMBER [specific number] p If I choose the second model to sweep, you should give me three parameters, the start frequency, the end frequency and the number of sample data. Three parameters are given in the form of the following keywords: START_F [specific number] Hz END_F [specific number] Hz SAMPLE_NUMBER [specific number] p If the user does not provide any specific number, you should make your own decisions and add an appropriate number.  I want to measure the impedance in the normal mode. Using the keywords. I want to measure the impedance in the sweep mode. Using the keywords.

**Task (Admin):** I want to measure the impedance in the sweep mode. Using the keywords.

**Response:** START\_F 100 Hz

END\_F 10 kHz

SAMPLE\_NUMBER 50 p

**Actuator:** Trying to connect to dev7502 on interface PCIe

Connected to dev7502 via data server 192.168.117.58:8004 and interface PCIe

Will perform 2 sweeps...

Individual sweep progress: 100.00%.

Returned sweeper data contains 2 sweeps.

sweep test finished!

After completing the execution, the data will be saved to a JSON file for subsequent analysis and processing. These data are presented in visual form in Figure S7B, which shows a standard 1,000 Ohm impedance test of two modes.

The subsequent data processing methods for the last two devices are similar; they are also saved as a unified JSON file, so further elaboration will not be provided in the following text.

### Device2: Broadband dielectric spectrometer

The broadband dielectric spectrometer is connected to the computer via GPIB-USB, and the interaction control is realized using the device's own GPIB commands (Figure S8).

#### System (Base Knowledge):

Version 1  Dielectric Testing	I need you to control the Alpha-A dielectric measurement instrument for conducting relevant tests. To complete the test, it is generally necessary to set 4 parameters, namely the sample diameter, the sample thickness, the starting frequency and the stopping frequency (scanning from high frequency to low frequency). Generally, the frequency sweep is from 1e7Hz to 1Hz. These four parameters should be given in the form of the following keywords: DIAMETER [specific number] mm THICKNESS [specific number] µm START_F [specific number] Hz END_F [specific number] Hz
-------------------------------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

I have a thin film sample with a diameter of 20mm and a thickness of 5.8µm. please help me conduct broadband dielectric testing.

**Task (Admin):** I have a thin film sample with a diameter of 20mm and a thickness of 5.8µm, and I want to terminate the frequency of 10Hz. please help me conduct broadband dielectric testing. Using the keywords.

**Response:** DIAMETER 20 mm THICKNESS 5.8 µm

START\_F 1e7 Hz END\_F 10 Hz

**Actuator:** The device is connected successfully.

Sweep frequency test begins.

Test finished after 136.1 s.

### Device3: System sourcemeter

During the measurement process of the leakage current parameters, two devices need to be controlled: one provides a high-voltage power supply, and the other performs the current measurement. The former, which was not mentioned in the main text, is the Keithley 2290-10[S9]. The high-voltage source is connected to the computer via GPIB-USB and is controlled using pure GPIB commands, while the microammeter is controlled through Test Script Processor (TSP®)[S10] instructions (Figure S9).

#### System (Base Knowledge):

<p>Version 1</p> <p>Leakage current measurement</p>	<p>I need you to control the voltage of the equipment to measure the leakage current of the film. Usually, the voltage measurement is a process of boosting the voltage from low to high, and the final voltage intensity is calculated based on the thickness of the film, which is approximately 100V/<math>\mu</math>m. When I need to perform the test, I need you to provide three parameters: the initial voltage, the final voltage, and the number of sample points (The voltage interval is preferably 30V). These three parameters should be given in the form of the following keywords:</p> <p>START_V [specific number] V      END_V [specific number] V      SAMPLE_NUM [specific number] p</p> <div style="background-color: #0072BD; color: white; padding: 5px; text-align: center;"> <p>I have a thin film sample with a thickness of 5.8<math>\mu</math>m. please help me conduct leakage current testing.</p> </div>
-----------------------------------------------------	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Task (Admin):** I have a thin film sample with a diameter of 20mm and a thickness of 5.8 $\mu$ m, and I want to conduct the leakage current test. please help me conduct leakage current testing.

**Response:** START\_V 0 V

END\_V 700 V

SAMPLE\_NUM 24 p

**Actuator:** Running Sweep . . .

Sweep Complete. Current Measurements:

[ -2.86102e-12, -1.01912e-10, -6.1512e-11, -6.88553e-11, -1.77538e-10, -1.79708e-10, -2.53987e-10, -2.31707e-10, -1.63257e-10, -2.50971e-10, -2.74646e-10, -4.36902e-10, -6.25837e-10, -1.82428e-09, 3.60382e-08, 7.34735e-10, 1.4087e-10, 7.39932e-11, 4.7493e-11, 3.08633e-11, 2.4271e-11, 2.21252e-11, 1.63555e-11, 1.36614e-11, 9.90629e-12 ]

Test finished after 76.8 s.

It is worth noting that the current system primarily focuses on the control of individual instruments, and samples still need to be manually placed. Therefore, it has not yet provided extensive artificial research assistance and device interaction capabilities. This is the direction that our future research can improve.

### Emergency scenarios in hardware control

During the actual operation of the equipment, some emergency situations may occur, such as hardware malfunctions, power disruptions, or communication errors. To ensure the safety of physical device control in these emergency situations, we took Device 1 as an example and conducted the following tests.

**Hardware malfunctions:** Since it is not possible to directly damage the equipment in the laboratory, we simulate a hardware fault by interrupting the measurement circuit during the testing process. Taking the interruption process of resistance measurement as an example, the initial measurements are of resistance, but later, due to the circuit being cut, it is equivalent to an open-circuit measurement. The measurement does not stop, but the data becomes abnormal, which can be identified during the later data analysis.

**Power disruptions:** We are making calls through an API, if the device loses power, communication with the device becomes impossible. At this point, an error will be reported in our Python code, and the program will be unable to run. However, this situation does not affect the device itself because it has already been powered off. The program will automatically abort when we cut off the power to the device during measurement.

**Communication errors:** It is almost the same as a power disruption, where communication cannot be established, except that one occurs during the measurement process and the other occurs before the

measurement begins. At this point, it is still necessary to manually troubleshoot and check if there are any communication issues.

It can be seen that our approach of pre-encapsulating the API for hardware control is relatively safe. The only thing to note is the reasonableness of the parameters generated by the LLM, such as the applied voltage. At this point, a check for parameter reasonableness will be added to ensure the normal operation of the equipment (essentially dealing with hallucinations).

## Note S7 - Examples of failure generations

Due to the hallucinations or lack of concentration caused by long memory chains in LLM, errors in the content generated by the system can occur both in stage 1 and stage 2. Here we list several typical errors in generated MATLAB code, including those that the system can automatically detect and fix.

**Missing parentheses (Automatic fix):** Most commonly seen in print results. The bug fixing strategy is to check if the parentheses inside the print match, and if there is a mismatch, it is usually done by adding a parenthesis before the square bracket.

System output	<pre>... disp(['Best Parameters: Number of Trees = ', num2str(best_params(1)), ', Min Leaf Size = ', num2str(best_params(2))]; disp(['Lowest RMSE: ', num2str(best_rmse)]); ... </pre>
Problem correction	<p>If there are parentheses in the contents of 'num2str', there is a great probability that an error will occur</p> <pre>... disp(['Best Parameters: Number of Trees = ', num2str(best_params(1)), ', Min Leaf Size = ', num2str(best_params(2))]); ... </pre>

**Return value failure:** The return value is Inf, NaN, or there is no return value in the MATLAB code. Strictly speaking, this is not a failure, but it is not conducive to the operation of the subsequent evaluation part. However, the occurrence of this situation means that there may be logical errors in the code generated by LLM, which resulted in the final result of its operation being NaN. For example, in the following output case, LLM used the 'str2double' function to convert 'y\_pred' to NaN in the end.

System output	<pre>... model = TreeBagger(numTrees, X_train, y_train, 'method', 'regression', 'MinLeafSize', minLeafSize);  % Make predictions for the test set y_pred = predict(model, X_test);  % Convert predictions to double y_pred = str2double(y_pred);  % Calculate RMSE for the test set rmse = sqrt(mean((y_test - y_pred).^2)); disp(['Root Mean Squared Error (RMSE) on test set: ', num2str(rmse)]); ... </pre>
---------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Undefined variables:** LLM may call variables that it may think have been defined in the program, but in reality, they are not defined. For example, xgboost below DMatrix is undefined, "Unable to resolve the name 'xgboost. DMatrix'."

System output	<pre> ... % Split the data train_days = days(1:60)'; train_C = C(1:60)'; test_days = days(61:70)'; test_C = C(61:70);  % Prepare data for xgboost dtrain = xgboost.DMatrix(train_days, label=train_C); dtest = xgboost.DMatrix(test_days, label=test_C); ... </pre>
---------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Dot indexing failure:** Dot indexing is not supported for variables of this type

System output	<pre> ... % Define the SVM regression model fun = @(x)svm_rmse(x, X_train, Y_train, X_test, Y_test);  % Define the optimization parameters opts = struct('MaxObjectiveEvaluations', 50, 'AcquisitionFunctionName', 'expected-improvement-plus');  % Perform Bayesian optimization to automatically tune SVM parameters best_params = bayesopt(fun, [1e-6, 100], opts, 'IsObjectiveDeterministic', true, 'MaxTime', 600, 'NumSeedPoints', 3, 'VariableDescriptions', {'BoxConstraint'}); ... </pre>
---------------	----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Incorrect dimensions:** A very common failure happened in matrix multiplication or function call. It can also be 'Arrays have incompatible sizes for this operation' in the following System output 2 ('y\_train - y\_pred').

System output 1	<pre> ... % Split the data into training (first 60 days) and test (remaining 10 days) sets X_train = days(1:60); y_train = C(1:60); X_test = days(61:end); y_test = C(61:end);  % Add lagged capacitance values as features num_lags = 5; % Number of lagged values to include X_train_lagged = zeros(length(X_train) - num_lags, num_lags + 1); for i = 1:num_lags     X_train_lagged(:, i) = y_train(num_lags - i + 1:end - i); end X_train_lagged(:, end) = ones(size(X_train_lagged, 1), 1);  % Train a linear regression model on the lagged features beta = regress(y_train(num_lags+1:end), X_train_lagged); ... </pre>
System output 2	<pre> ... % Split the data into training (first 60 days) and test (remaining 10 days) sets X_train = days(1:60); y_train = C(1:60); X_test = days(61:end); y_test = C(61:end); </pre>

```

% Initialize parameters
num_trees = 1000;
max_depth = 10;
learning_rate = 0.1;

% Initialize predictions using the average of the training samples
y_pred = mean(y_train) * ones(size(y_test));

for i = 1:num_trees
    % Compute the residuals by subtracting the predictions from the actual values
    residuals = y_train - y_pred;
...

```

**Miscalculation:** The execution section will return a result value, but it deviates significantly from the actual value. This indicates that there is an error in the logic during the calculation, such as an incorrect calculation of RMSE.

Even equipped with the state-of-the-art LLM as GPT-4o, the probability of these errors occurring is also not low (38% error rate for GPT-3.5-turbo, while the error rate of GPT-4o may be slightly lower). Generally, it is necessary to combine many other technologies to improve accuracy, and our system adopts LLM that have not been finely tuned, so it is inevitable to have a relatively high error rate. As shown in the optimization results presented in the main text, although these errors are not enough to have a significant impact on the final result, they still have some interference with system performance. Therefore, in the future, it is recommended to use more tools to further improve the accuracy of LLM, thereby enhancing the efficiency of the whole system.

## Note S8 - Analysis of ablation Studies and hyperparameters

In our LLM-based system, there are many variables affecting the final output or the whole performance of the system, so to further investigate the impact we conduct some ablation studies. The factors explored here are not entirely sufficient, but a rough understanding of the influencing factors can be seen.

In the main text, we have made a brief comparison between using a complete program and only using a functional framework. The specific accuracy scoring rules are shown in Table S2. Other scoring rules, such as time being a relative value, with a minimum time of 5 and a maximum time of 1. This is actually a type of ablation experiment used to control variables to evaluate the influencing factors of the system, and the same applies to subsequent hyperparameter comparisons.

It is important to note that the above table is mainly for scoring the functions generated by LLM in the first stage iteration. If the operation of stage 1 is incomplete (Iteration aborted or timed out due to errors, etc.), the score will be less than or equal to 2, while complete operation will be greater than or equal to 3. Although these descriptions are subjective textual expressions, their evaluation should be relatively objective. We will run the complete code generation and function code generation programs separately for a certain number of times, and score them according to this table based on their generation. In some literature, a similar approach has actually been adopted, where they either directly score or are scored by experts[S11][S12].

All hyperparameters used by the Repower are listed in Table S3, among which the program execution time in stage 1&2 can be adjusted according to the difficulty of the task (for example we list two types of time we used in stage 2: 900s and 1200s). In Stage 2, the number of initial function groups is the number of selected first-stage functions (each function enters a function group individually) plus additional blank function groups. The purpose of adding extra blank function groups has been mentioned in the main text, which is to store outliers and maintain evolutionary diversity. We basically use the original parameters for LLM and haven't made too many adjustments including the temperature. As a general rule, different methods of prompt should be selected according to the different methods at the same temperature, but we have made it a unified form, which may be an area for future optimization. Restarting and backtracking are

both small strategies to avoid LLM getting stuck in a fixed mindset. Restarting refers to resetting the context of the Check agent to its initial value after a certain length of context has passed, while backtracking involves removing error messages and generated content from the main program if restarted many times (Restart limit of Check agent).

It must be admitted that the ablation experiment in this article was not well done, such as not considering the impact of timeout time and large model types on the generated results (although it may not be important). The exploration of LLM may be limited to simple direct calls without using extension tools such as LangChain[S13] or Retrieval-Augmented Generation (RAG)[S14], which can greatly improve work efficiency or overall system performance. If eager to gain a deeper understanding of the operating mechanism of this system, we should further attempt to analyze the impact of each parameter on the generation process rather than just the results in the future, which may help scholars better understand the operating mechanism of LLM agents.

## Note S9 - Method searching process of the three tasks

In the main text, we have already learned that due to the different characteristics of data types across various tasks, LLMs will ultimately evolve different methods. In fact, in the first stage of method search, different tasks exhibit distinct features. Here, the method exploration processes for the three tasks mentioned in the main text are briefly described as follows:

For Task 1, the majority of the search process in the first stage will likely progress from LR to SVR, then to regression trees, and finally to methods that consider time series, such as LSTM. This is because Task 1 is essentially a sequence prediction problem involving the change in capacitance values over time.

For Task 2, the LLM mostly still attempts LR first, followed by SVR, and then various types of regression tree models such as GBR and RF. Perhaps regression tree models can better capture the relevant features of this type of optimization problem.

For Task 3, in the first stage of the search, the LLM is likely to try SVR first, then regression tree models, and finally neural networks. This is because the input dimension of Task 3 is relatively large, thus requiring more complex models to extract features.

During the search process of the methods, some methods may appear repeatedly. However, in reality, their internal calculation methods or hyperparameters have been adjusted. Such changes are not reflected here. For detailed information, please refer to the experimental data in the source code attachment under <https://zenodo.org/records/14216501>.

## Note S10 - Comparison with human baselines

It is difficult to find human baselines for comparison due to the lack of standard answers to the problems involved in our system. In the main text, we compared the results of the first task with those in a literature, but this is not a comprehensive comparison either. In order to further demonstrate the effectiveness of the system in improving research productivity, we have decided to limit the scope of comparison to using LLM to complete research tasks. The first mode of comparison is manual, which means that people only use the LLM on the webpage (ChatGPT 3.5) to complete the first round of related optimization tasks through dialogue, and prompt words are given by people's independent thinking. The second mode is semi-automatic, where humans rely on our system's automatic execution module but prompt words are provided by humans. The third mode is fully automatic, and all steps are handed over to our system to complete. The three types can be seen as a process of self-comparison, where certain parts of our system are replaced by humans, and the results are presented in Figure S13.

Due to the randomness of the LLM generation method, it can be observed that the RMSE in automatic mode may not necessarily be superior to the first two modes, but only performs more prominently in Task 2. In terms of time cost, automatic mode incurs significant overhead on more complex tasks (Task 2 and Task 3), with most of its time coming from the autonomous search process of hyperparameters, which is not included in manual and semi-automatic modes (Considering the limited time available for human involvement, neither of these two modes requires the LLM to perform hyperparameter search, which conversely highlights the convenience brought by automation). If we merely compare the optimization time for ordinary search iterations, the automatic mode would surpass the first two modes. The manual mode does not incur any cash expenses because of the direct use of web side conversations, and the expenses

for the latter two are relatively low, reaching a maximum of \$0.08. It is worth noting that there is no direct correlation between cash expenditure and time cost, as most of the time cost is spent on program execution rather than code generation.

Through the above comparison and analysis, it seems that the comprehensive performance of semi-automatic mode and manual mode is better than that of automatic mode. In fact, the prompt words in automatic mode are often not as deep and flexible as human thinking, so it will to some extent affect the generation of LLM. However, the testing of the first two modes requires the person to always be in a working state, while the automatic mode can save this time for people to deal with other matters. The time cost of Stage 1 shown here may have reached nearly an hour for some workers, while Stage 2 is likely 10-20 times that, which is usually unacceptable for continuous human work. If human rest time is added, its time cost is also close to that of automatic mode. In addition, although manual mode has no additional costs, labor costs may also far exceed machine costs. So there is still some significance in developing this system.

In addition to the above comparison methods, open-source information such as Google, arXiv and Wiki can be used as our human baselines. Because parameter optimization also has processes and steps. By comparing the completeness and accuracy of the process steps, we can also compare the ideas given by LLMs with the existing human parameter adjustment method. However, we believe that this comparison method may not be fair because the method steps in these literatures have been formed through many iterations of previous knowledge or thinking, and the time and cash spent may far exceed the resources required for LLM thinking. Therefore, we did not adopt this comparison method.

## Note S11 - Initial design of the system

The whole system is developed in Jupyter Notebook. In order to present the same Interface display as the ChatGPT as much as possible, we add a lot of print functions to the code but this will make the system run slowly (Figure S14). Furthermore, it is not recommended to operate the system in such a unified front and rear user interface, instead, we should rebuild our own front-end interface. As a matter of fact, our system is just the prototype of an engineering project, and there are still many details that need improvement.

Perhaps someone has noticed that the framework or workflow of our system is similar to reinforcement learning, and indeed it is, because it is almost impossible to bypass such a working paradigm of machine learning when solving complex problems with LLM. For example, the Reflexion[S19] method used for improving the accuracy and overall performance of LLM has a curious coincidence with reinforcement learning after a carefully analysis of the framework of both. We think that the best working mode is reinforcement learning combined with LLM self-optimization, but there are many theoretical problems in it, such as how to define reward and action space, especially the action space seems infinite and difficult to be mathematised in our problem. As mentioned in the main text, these terms can be vividly described but are difficult to define accurately. Therefore, we adopted a more eclectic approach in our research, that is to separate this part from LLM like most of the scholars did.

Our goal is to optimize the effect of the algorithm through workflow. If the effect of the algorithm is improved after several iterations, we can exit. Our current goal may not be to kill all the prediction methods in the market, but to tell you that this workflow can indeed make the algorithm evolve, and the prediction effect has been amazing.

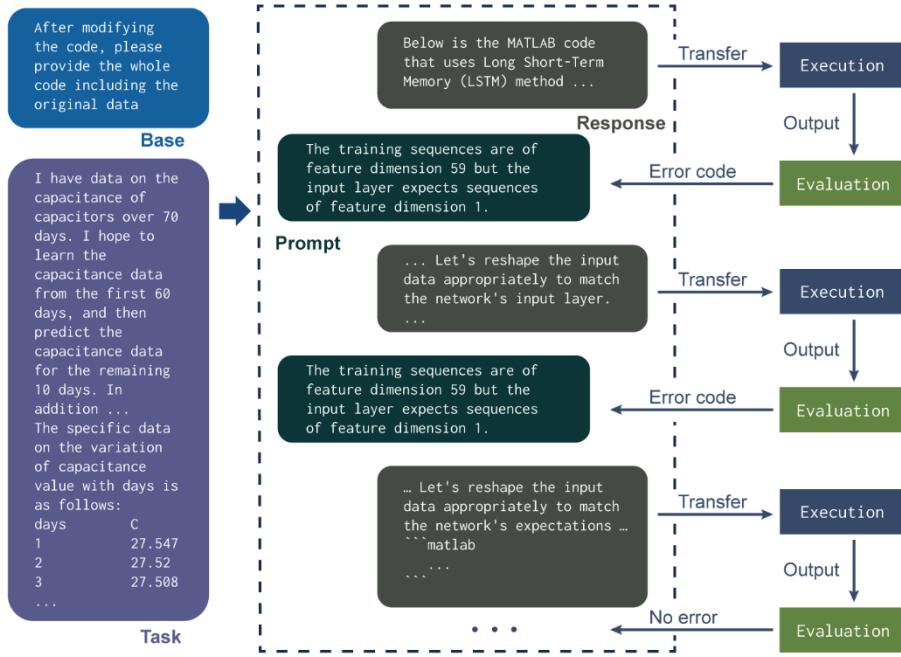
In our initial design, we let LLM generate all the operation codes including the original data, but due to its "hallucination" effect, these repetitive generation (such as the copy of the original data, the calculation process of RMSE, etc.) will occasionally make errors. To avoid this situation, we have improved the entire operation process and provided a code framework that only allows LLM to generate core prediction code. The remaining duplicate codes will be given in advance to avoid LLM performing repeated generation steps. Such a code framework can be given by domain experts or generated by LLM if researchers' knowledge in this field is insufficient or they are just novices.

## Note S12 - Other Notes

From the results of our experiment, applying a two-stage optimization strategy can indeed continuously optimize the results, but we can also briefly elaborate on the theoretical basis for this optimization method.

Taking the second stage of exerting main functions as an example, we construct the optimal combination of prompts based on the function pool and input them to LLM to generate new samples. In this continuous loop process, the parameters of LLM did not change, only the function pool and the prompts constructed from it changed. Since LLM is unknown to data, it can be seen as a black box attempt by LLM on the environment, which achieves results optimization by obtaining optimal feedback through trial and error. In this process, we utilized the function pool to help LLM obtain better feedback and encourage it to think deeply, which can help LLM better understand the unknown[S15]. If further compared to human exploration of the unknown world, the process is almost identical. Whether it is a new discovery, new cognition, or new creation, as long as it is new, it can be seen as feedback obtained from the black box and scientific discovery. In this regard, LLM and humans are consistent, and perhaps the emergence of artificial general intelligence[S16] in the future is not far away.

## B Supplemental Figures



### **Figure S1. Example workflow of the Stage 1**

The goal is set to be fixing errors in the code after the initial generation.

**A**

```

Task 1

1 function avgRMSE = prediction(x_train, y_train, x_test, y_test)
2     num_days = size(y_test, 1);
3     num_capacitors = size(y_test, 2);
4
5     predicted_values = zeros(num_days, num_capacitors);
6     rmse_values = zeros(1, num_capacitors);
7
8     for i = 1:num_capacitors
9         % Feature scaling
10        x_train_scaled = zscore(x_train);
11        x_test_scaled = zscore(x_test);
12
13        % Support Vector Machine (SVM) hyperparameter tuning with k-fold cross-validation
14        svm_model = fitrsvm(x_train_scaled, y_train(:, i), 'OptimizeHyperparameters', 'all', ...
15            'HyperparameterOptimizationOptions', struct('AcquisitionFunctionName',...
16            'expected-improvement-plus', 'KFold', 5));
17        svm_predicted_values = predict(svm_model, x_test_scaled);
18        rmse_values_svm = calculateRMSE(svm_predicted_values, y_test(:, i));
19
20        % Polynomial Regression hyperparameter tuning with k-fold cross-validation
21        best_deg = 1;
22        best_rmse_poly = inf;
23
24        for degree = 1:min(5, size(x_train, 1)-i)
25            mdl = fitlm(x_train, y_train(:, i), ['poly', num2str(degree)]);
26            poly_predicted_values = predict(mdl, x_test);
27            rmse_poly = calculateRMSE(poly_predicted_values, y_test(:, i));
28
29            if rmse_poly < best_rmse_poly
30                best_rmse_poly = rmse_poly;
31                best_deg = degree;
32                predicted_values(:, i) = poly_predicted_values;
33            end
34        end
35
36        % Ensemble Method: Stacking
37        stacked_data = [zscore(svm_predicted_values), zscore(poly_predicted_values)];
38
39        % Combine predictions from SVM and Polynomial Regression models using linear regression
40        mdl_stack = fitlm(stacked_data, y_test(:, i));
41        stack_predicted_values = predict(mdl_stack, stacked_data);
42        rmse_stack = calculateRMSE(stack_predicted_values, y_test(:, i));
43
44        if rmse_stack < min(rmse_values_svm, best_rmse_poly)
45            predicted_values(:, i) = stack_predicted_values;
46            rmse_values(i) = rmse_stack;
47            disp(['Stacking of SVM and Polynomial Regression selected for Capacitor ', num2str(i)]);
48        elseif rmse_values_svm <= best_rmse_poly
49            predicted_values(:, i) = svm_predicted_values;
50            rmse_values(i) = rmse_values_svm;
51            disp(['SVM selected for Capacitor ', num2str(i)]);
52        else
53            rmse_values(i) = best_rmse_poly;
54            disp(['Polynomial Regression with degree ', num2str(best_deg), ...
55                  ' selected for Capacitor ', num2str(i)]);
56        end
57    end
58
59    avgRMSE = mean(rmse_values); % Calculate average RMSE over all capacitors
60
61    % Display RMSE for each capacitor
62    disp('RMSE for each capacitor:');
63    disp(rmse_values);
64 end

```

**B**

```

Task 2

1 % Principal Component Analysis (PCA) for feature selection
2 [coeff, ~, ~, ~, explained] = pca(X_scaled);
3 total_explained = cumsum(explained);
4 num_features = find(total_explained >= 99, 1);
5 X_train_pca = X_scaled * coeff(:,1:num_features);
6 % Cross-validated hyperparameter optimization
7 % with Bayesian optimization and grid search
8 cv = cvpartition(size(X_train_pca, 1), 'Kfold', 5);
9 opt = struct('Optimizer', 'bayesopt', 'ShowPlots', ...
10 true, 'CVPartition', cv, 'AcquisitionFunctionName', ...
11 'expected-improvement-plus');
12 t_max = templateTree('MaxNumSplits', 100);
13 t_min = templateTree('MaxNumSplits', 100);
14 % Increase the number of learners in the ensemble for better performance
15 n_learners = 500;
16 mdl_max = fitrensemble(...)
```

**C**

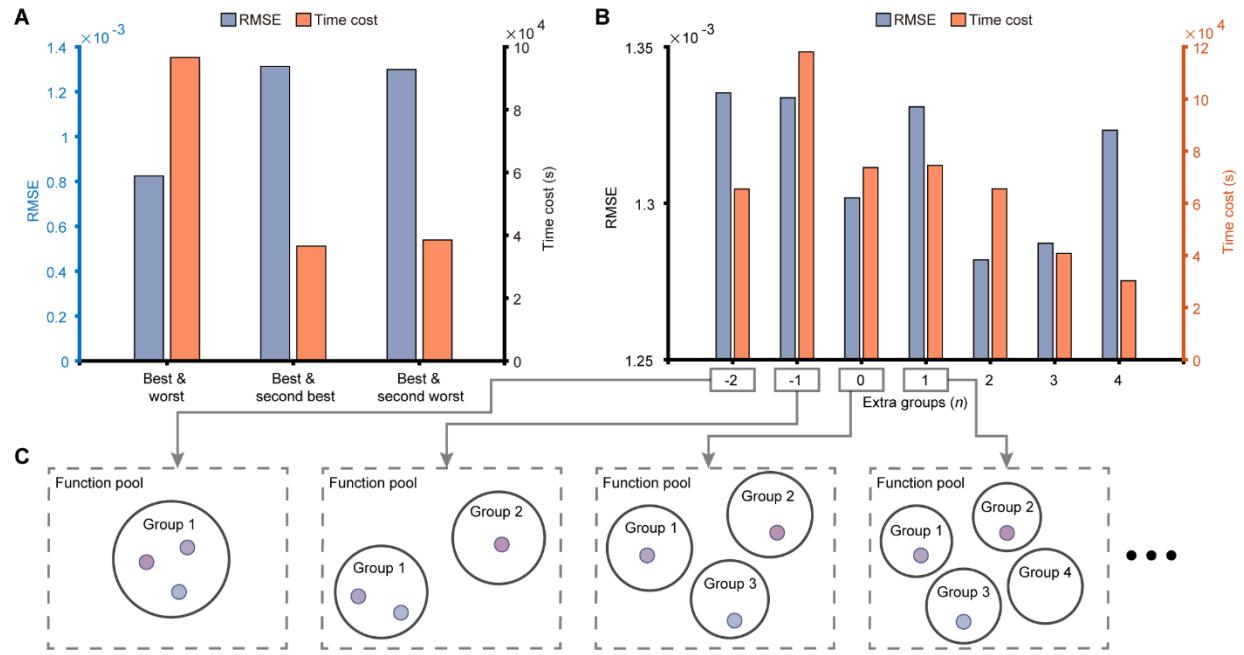
```

Task 3

1 % Introduce enhanced feature engineering with polynomial features
2 % Set the degree of the polynomial features to 3 for more complexity
3 poly_degree = 3;
4 X_train_poly = [] ; X_test_poly = [];
5 % Create polynomial features up to degree 3
6 for d = 1:poly_degree
7     X_train_poly = [X_train_poly, X_train_normalized.^d];
8     X_test_poly = [X_test_poly, X_test_normalized.^d];
9 end
10 % Scale the output variables using z-score normalization
11 mu_C = mean(C_train); std_C = std(C_train);
12 scaled_C_train = (C_train - mu_C) / std_C;
13 % Train a neural network regression model to predict scaled capacitance (C)
14 % and equivalent series resistance (ESR) using all polynomial features
15 model_C = fitrnet(X_train_poly, scaled_C_train);
16 model_ESR = fitrnet(X_train_poly, scaled_ESR_train);
```

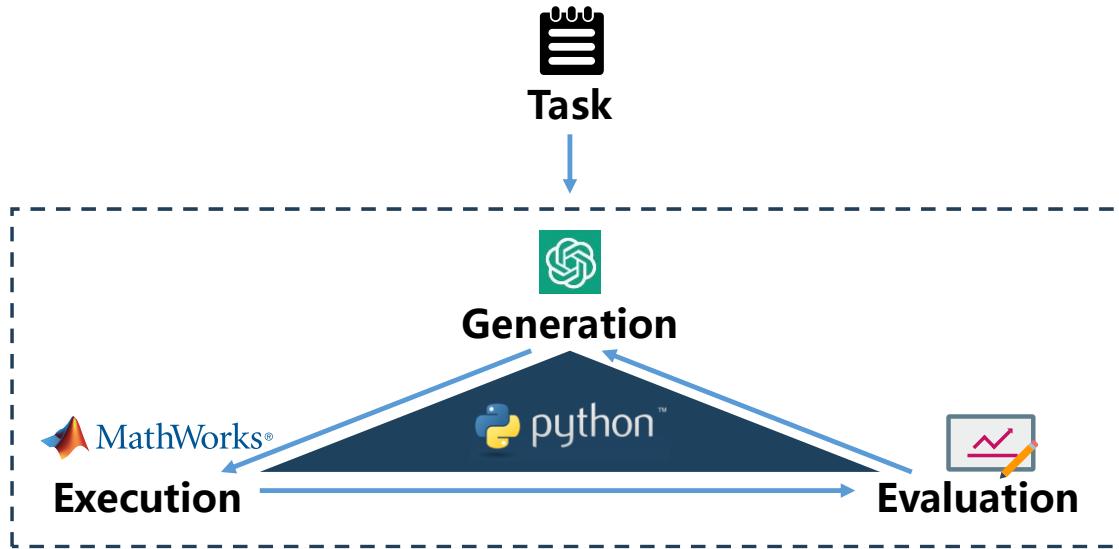
**Figure S2. The final output examples of three tasks after two-stage search**

(A) Task 1 final output. The method used is a combination of linear regression and SVM. (B) Task 2 final output. Due to the relatively long length of the generated code, only the key parts are presented for tasks 2 and 3. (C) Task 3 final output.



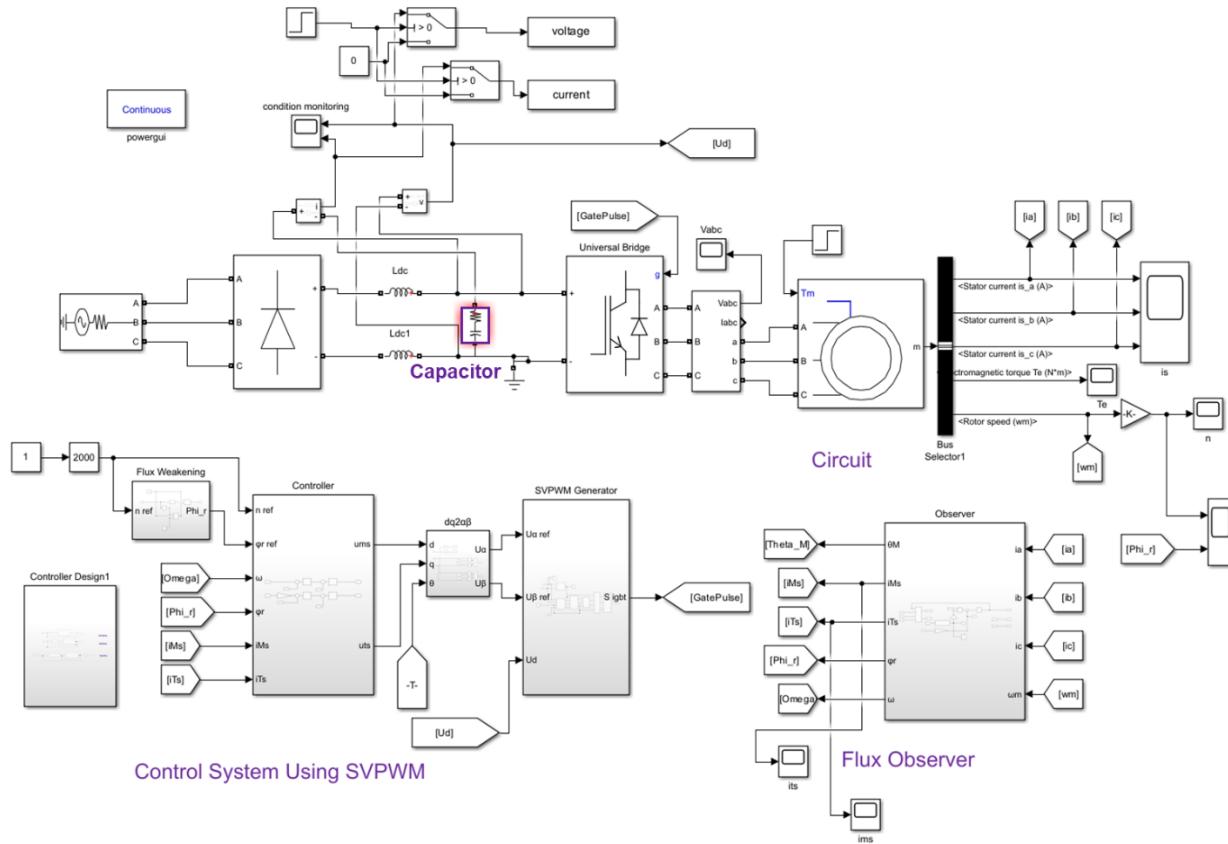
**Figure S3. Exploration of other performance factors**

**(A)** Comparison of evolution results of Stage 2 prompt constructions. The result presents the final output of the whole code generation for Task 2, and the same applies to Figure B. The method used is a combination of linear regression and SVM. **(B)** Comparison of evolution results of Stage 2 extra groups. Taking the number of functions entering the second stage,  $k=3$ , as an example, the range of  $n$  varies from -2 to 4. **c**, Diagram of the extra group within the function pool.



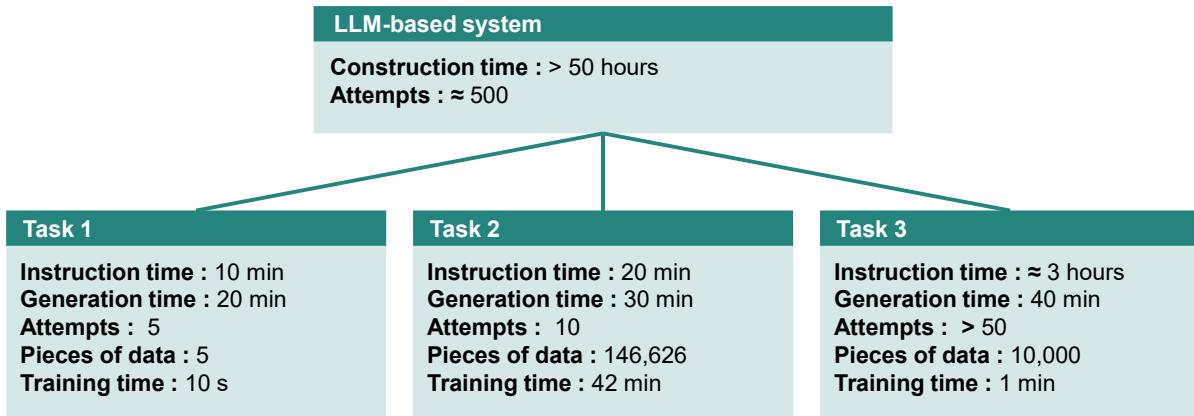
**Figure S4. System architecture diagram**

In the dotted box are automated operations. The Python triangle in the middle is the transport layer, responsible for data transmission between different parts. There is no distinction between Stage 1 and Stage 2 here, as both stages follow this cyclic process.



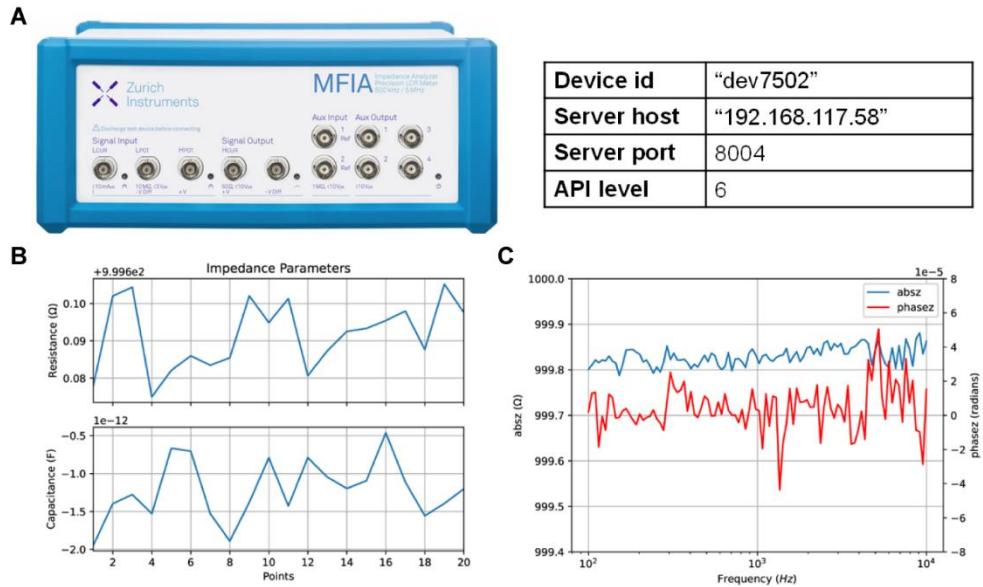
**Figure S5. Condition monitoring simulation diagram**

Simulation component diagram in Simulink. The purple box contains the capacitor used for condition monitoring.



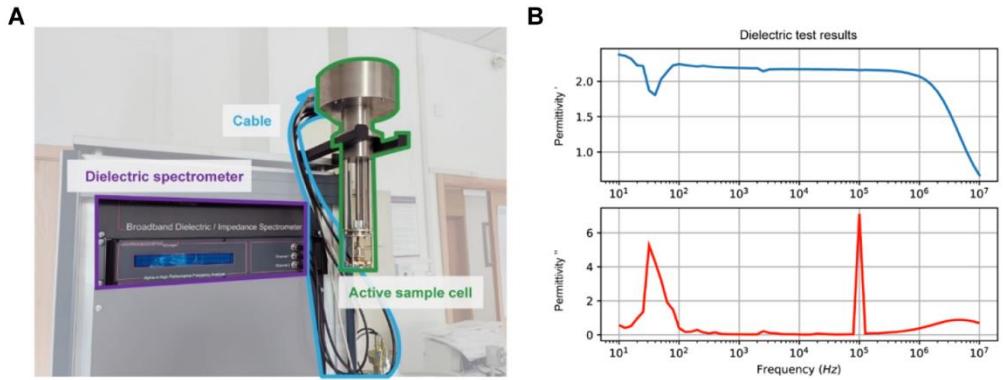
**Figure S6. Part of the detailed information for building the system and completing tasks**

The instruction time of each task means the time spent to come up with a clear instruction which can get the correct output, namely the time to generate a suitable prompt by human. The generation time is the average operation time in the first stage using our LLM-based system. Attempts are number of prompt attempts to instruct the LLM. Pieces of data are the amount of data obtained in advance. Training time is the running time of an entire code which contains all pieces of data, but usually not all data are used for training in order to improve efficiency.



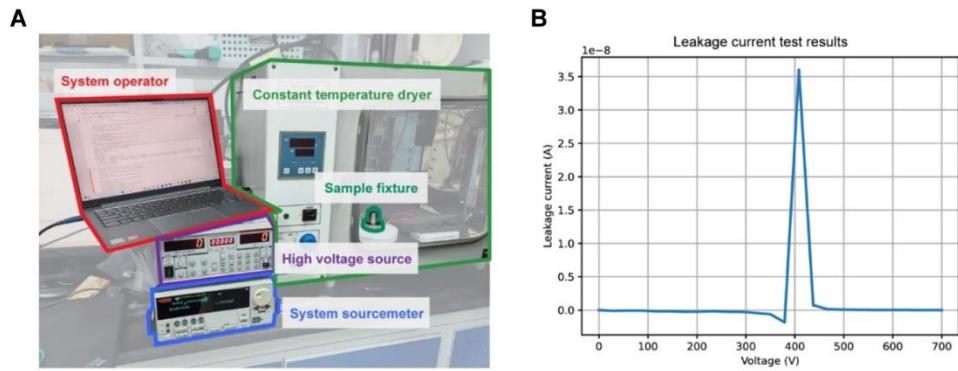
### **Figure S7. Impedance test equipment and test results**

(A) Equipment and connection set.. (B) Ordinary impedance test results. (C) Sweep frequency test results.



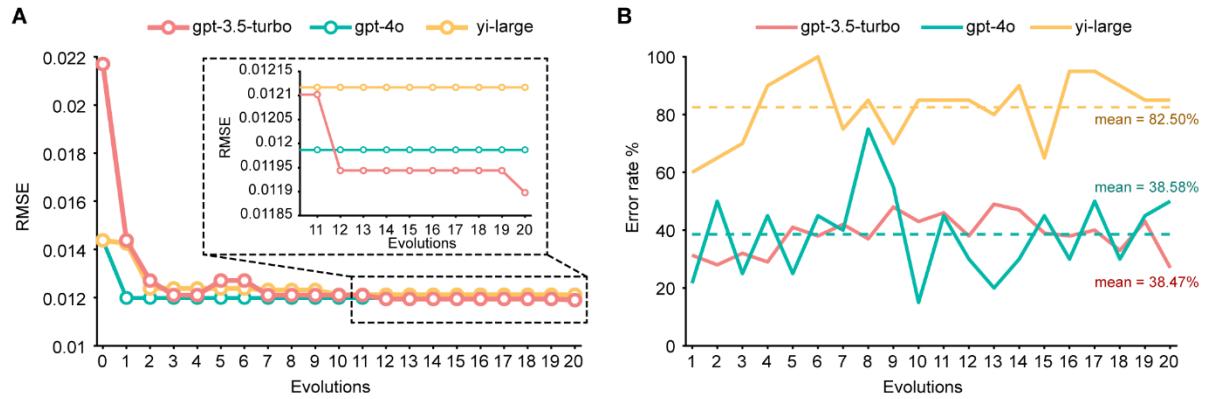
**Figure S8. Dielectric test equipment and test results**

(A) Broadband dielectric spectrometer physical connection. The computer is connected to the instrument via the GPIB interface at the rear. (B) Dielectric test results.



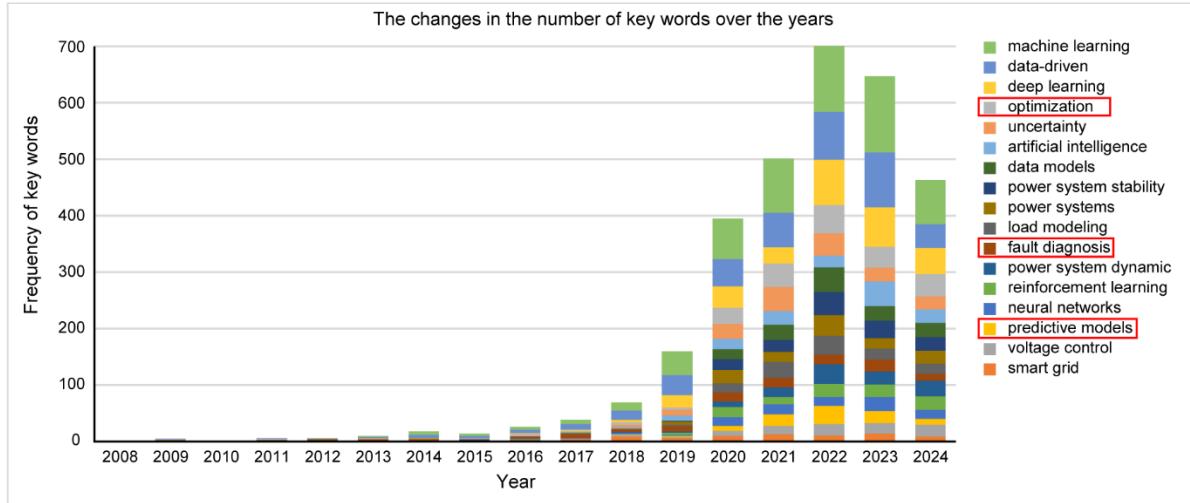
**Figure S9. Leakage current test equipment and test results**

(A) System sourcemeter physical connection. (B) Leakage current measurement results.



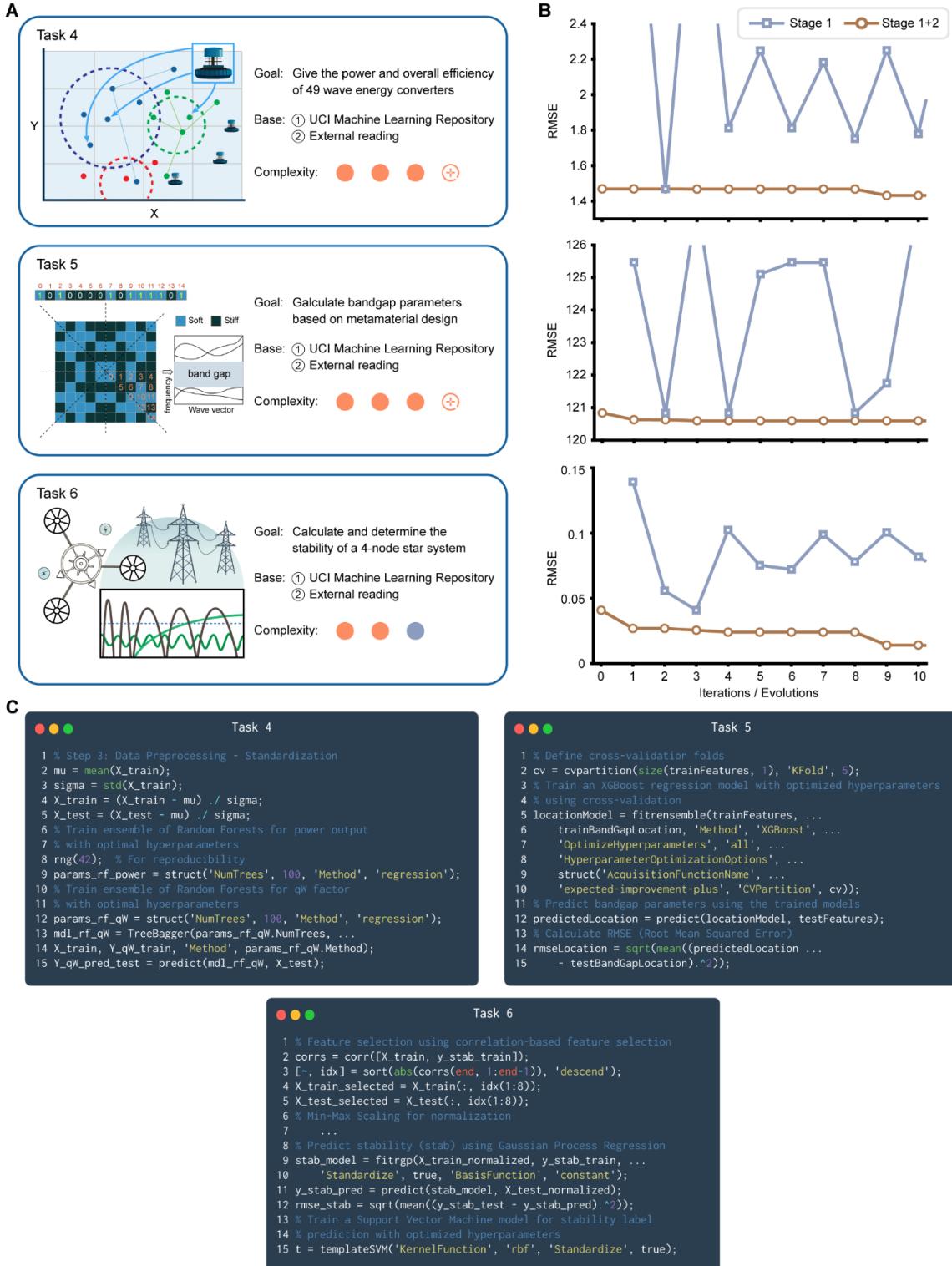
**Figure S10. The performance comparison of different large models in Task 1**

(A) RMSE comparison. The results are all in the function generation pattern. (B) Average error rate comparison.



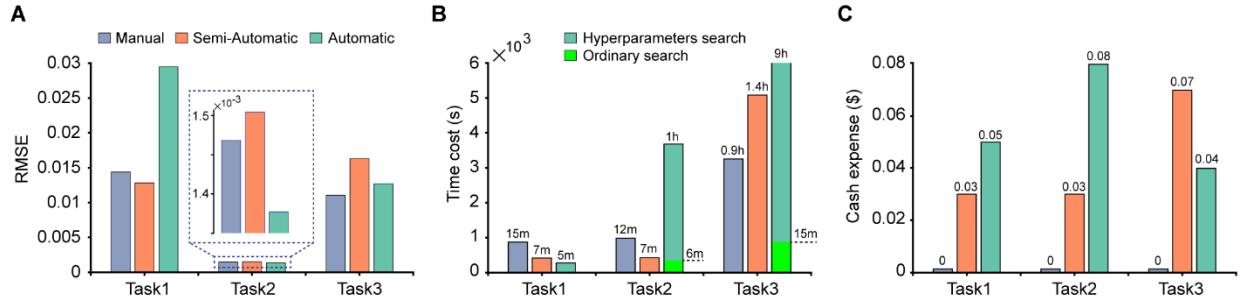
**Figure S11. The annual changes of keywords in the literature of data-driven in power system**

A total of 17 keywords were extracted from the 6,194 relevant literatures retrieved from Web of Science, among which the keywords circled in the red box are the tasks we presented in the main text (fault diagnosis and state estimation are almost closely related).



**Figure S12. Task results comparison using different evolutionary frameworks**

(A) The evolutionary process comparison of the three frameworks. The result of RePower is the second stage evolutionary process that connects the first stage. (B) The final result comparison of the three frameworks. (C) The final output examples of the tasks



**Figure S13. Comparison with human baselines**

These bar charts show the RMSE performance, time cost, and cash expense of three completion modes (manual, semi-automatic, and fully automatic, which is described detailed above) in Stage 1 iterations, respectively. **(A)** RMSE performance comparison. **(B)** Time cost comparison. **(C)** Cash expense comparison.

In this code:

1. We load the capacitance data into variables 'days' and 'C'.
2. We split the data into training and testing sets.
3. We train a linear regression model using the training data.
4. We use the trained model to predict the capacitance values for the testing days.
5. We calculate the RMSE between the predicted values and the actual values for the testing days.
6. Finally, we display the RMSE as a measure of the prediction performance.

You can run this code in MATLAB to get the RMSE value for the linear regression model's predictions on the remaining 10 days of capacitance data.

**MATLAB\_Assistant:** mes = no error

res\_1 = Root Mean Squared Error (RMSE): 0.034012

**User:** The current RMSE is 0.034012. I want to make the RMSE lower than 0.01. Try to use another prediction method and add the automated parameter searching step.

**Response:** To improve the prediction accuracy and achieve an RMSE lower than 0.01, we can try using a Support Vector Machine (SVM) regression model along with an automated parameter optimization process. We can use grid search along with cross-validation to find the best parameters for the SVM model.

Here is the updated MATLAB code with SVM regression and automated parameter optimization:

**Figure S14. System output in the Jupyter Notebook**

## C Supplemental Tables

**Table S1. The input and output data dimensions of the five tasks**

Task	Input data meaning	Input Data dimension	Output Data dimension
1	Capacitance	60×1	10
2	Optimization parameters	9×1	2
3	Voltage and current	101×2	2
4	X/Y coordinate	49×2	49+1
5	Binary code	14×1	2
6	Control parameters	3×4	1+1 (classification)

**Table S2. Accuracy Scoring rules for LLM generating programs**

Accuracy score	meaning
1	Incorrect generation including missing key characters or timeout
2	Has a tendency to solve problems but generates simple errors. For example, incorrect use of operator. (Incomplete operation of stage 1)
3	Correct generation without execution error, but distort the requirements of the task. (complete operation of stage 1)
4	Correct generation without execution error, fully resolve the problem but the result is not satisfied.
5	Correct generation without execution error, fully resolve the problem and the result is within a relatively ideal range.

**Table S3. Hyperparameters of RePower**

	Hyperparameters	value or setting
<b>LLM</b>	Model	GPT-3.5-turbo-0125
	Temperature	1.0 (default)
<b>Horizontal search (Stage 1)</b>	Iterations	10
	Max context length of Check agent	5
	Restart limit of Check agent	3
	Backtrack limit of Info agent	5
	Generation type	function/whole code
	Program execution time limit	900s/36000s
<b>Vertical search (Stage 2)</b>	Number of initial functions ( $k$ )	3
	Number of extra groups ( $n$ )	2
	Number of initial groups	$k + n = 5$
	Prompt construction	Best and worst
	Context length limit of agent	3
	Iterations of each evolution	20
	Max evolutions	20
	Program execution time limit	900s/1200s

**Table S4. Method abbreviation in the main text**

Method	Meaning	Method	Meaning
LR	Linear Regression	LSTM	Long Short Term Memory Network
PR	Polynomial Regression	MLP	Multilayer Perceptron
Ridge	Ridge Regression	ANN	Artificial Neural Network
Lasso	Lasso Regression	CNN	Convolutional Neural Network
ARIMA	Autoregressive Integrated Moving Average model	AdaBoost	Adaptive Boosting
RF	Random Forest	CatBoost	Categorical Boosting
Trees	Decision Trees	LSBoost	Least-square Boosting
CART	Classification and Regression Tree	LightGBM	Light Gradient-Boosting Machine
GPR	Gaussian Process Regression	XGBoost	eXtreme Gradient Boosting
GBR	Gradient Boosting Regression	SVR	Support Vector Regression
Bayesian	Perform Bayesian Optimization to find the best hyperparameters	SVR RBF	SVM with Radial Basis Function Kernel

## D Supplemental Methods

### Theoretical justification for two-stage architecture.

The system we have built is essentially an intelligent agent designed to solve tasks. In the realm of LLMs, an agent refers to the deployment of LLMs to perform complex tasks traditionally handled by humans, leveraging their capabilities for intelligent analysis, decision-making, and execution[S17]. Research in this field focuses on designing implementation processes that effectively integrate LLMs into specific tasks, defining their operational modes, workflows, and the step-by-step functioning of the entire system. For instance, ReAct[S18] combines reasoning and action in sequential rounds to enhance task execution. Feedback, crucial in both control systems and human processes, has also been integrated into LLM workflows. Reflexion[S19], a notable example, employs feedback mechanisms to help LLMs process context information, using a multi-agent structure for reflective reasoning that draws on reinforcement learning principles. In the both stages of our system, our approach follows a similar concept: generating predictive code where the environment comprises all data, observed values are prediction errors, and strategies are generated by the LLM based on feedback.

Inspired by Cumulative Reasoning[S20], fast and slow thinking[S21] and evolutionary search[S22]-[S24], aforementioned theories and our experimental exploration, we believe that the success of the two-stage framework depends on fast thinking based on reflection and slow thinking based on evolution. The first stage of the search is rapid, where the LLM is not prompted to engage in extensive deliberation but instead directly switches to the next method based on the current error value. Compared to the second stage, the search time in the first stage is relatively shorter (taking one search result of Task 1 as an example, the first stage took 1968.8355 seconds, while the second stage took 6322.3715 seconds). The aim is to roughly scan through the existing methods, and our results indicate that the LLM indeed performs in this manner, as illustrated in Figure 3D.

The second stage is slow thinking, which requires the LLM to delve into the potential for enhancing the methods from the first stage. This involves updating the feature selection of the methods and fine-tuning hyperparameters through a population-based evolutionary approach, thereby achieving algorithmic evolution. For instance, in Task 2, as the number of evolutionary iterations increases, the LLM learns to utilize principal component analysis (PCA) for dimensionality reduction and feature selection, followed by Bayesian optimization for hyperparameter tuning and regression trees for prediction.

We allow the LLM to engage in rapid, intuitive thinking to explore various methods in the first stage, followed by slow, meticulous thinking to achieve algorithmic evolution in the second stage, thereby endowing the entire system with the logical reasoning capabilities necessary to solve complex problems. As for the relevant theories of the evolutionary process, we have elaborated on them in the “Evolution Strategy” section of the Methods. In summary, the two-stage collaboration enables the LLM to accomplish the tasks we have set, thereby achieving the objectives of the agents.

## E Supplemental References

- [S1] Mathworks (2023). Call MATLAB from Python - MATLAB & Simulink. [www.mathworks.com](http://www.mathworks.com). <https://www.mathworks.com/help/matlab/matlab-engine-for-python.html>.
- [S2] Anaconda Inc. (2018). Anaconda. Anaconda. <https://www.anaconda.com>.
- [S3] Ritamaki, M., Rytoluo, I., and Lahti, K. (2019). Performance metrics for a modern BOPP capacitor film. *IEEE Transactions on Dielectrics and Electrical Insulation* 26, 1229–1237. <https://doi.org/10.1109/tdei.2019.007970>.
- [S4] Zhang, Y.-X., Feng, Q.-K., Chen, F.-Y., Liu, D.-F., Pei, J.-Y., Zhong, S.-L., Yang, Z., and Dang, Z.-M. (2022). Long-term capacitance variation characteristics, law extraction, single and collaborative prediction of film capacitors at room temperature and humidity. *Microelectronics Reliability* 139, 114845–114845. <https://doi.org/10.1016/j.microrel.2022.114845>.
- [S5] Shu, Z., Tang, J., Guo Yuhua, and Lian, J. (2007). An Efficient SVPWM Algorithm With Low Computational Overhead for Three-Phase Inverters. *IEEE Transactions on Power Electronics* 22, 1797–1805. <https://doi.org/10.1109/tpel.2007.904228>.
- [S6] Neshat, M., Alexander, B., Sergienko, N., and Wagner, M. (2023). Large-scale Wave Energy Farm. UCI Machine Learning Repository. <https://doi.org/10.24432/C5GG7Q>.
- [S7] Ogren, A., Chen, Z., Brinson, L.C., Bastawrous, M., Rudin, C., and Daraio, C. (2021). 2D elastodynamic metamaterials. UCI Machine Learning Repository. <https://doi.org/10.24432/C5ZS5D>.
- [S8] Vadim, A. (2018). Electrical Grid Stability Simulated Data. UCI Machine Learning Repository. <https://doi.org/10.24432/C5PG66>.
- [S9] Tektronix (2024). Keithley 2290 Series. Tek.com. <https://www.tek.com/en/products/dc-power-supplies/2290-series>.
- [S10] Tektronix (2024). How to Write Scripts for Test Script Processing. Tek.com. [https://www.tek.com/en/documents/application-note/how-to-write-scripts-for-test-script-processing-\(tsp\)](https://www.tek.com/en/documents/application-note/how-to-write-scripts-for-test-script-processing-(tsp)).
- [S11] Boiko, D.A., MacKnight, R., Kline, B., and Gomes, G. (2023). Autonomous chemical research with large language models. *Nature* 624, 570–578. <https://doi.org/10.1038/s41586-023-06792-0>.
- [S12] Bran, A.M., Cox, S., Schilter, O., Baldassari, C., White, A.D., and Schwaller, P. (2024). Augmenting large language models with chemistry tools. *Nature machine intelligence* 6, 525–535. <https://doi.org/10.1038/s42256-024-00832-8>.
- [S13] Chase, H. (2022). LangChain. GitHub. <https://github.com/hwchase17/langchain>.
- [S14] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., et al. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In Proceedings of the 34th International Conference on Neural Information Processing Systems (NeurIPS) (Curran Associates, Inc.), pp. 9459–9474.
- [S15] Jia, M., Cui, Z., and Hug, G. (2024). Enabling Large Language Models to Perform Power System Simulations with Previously Unseen Tools: A Case of Daline. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2406.17215>.
- [S16] Goertzel, B. (2014). Artificial General Intelligence: Concept, State of the Art, and Future Prospects. *Journal of Artificial General Intelligence* 5, 1–48. <https://doi.org/10.2478/jagi-2014-0001>.
- [S17] Google (2025). Agents. Kaggle.com. <https://www.kaggle.com/whitepaper-agents>.
- [S18] Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. (2023). ReAct: Synergizing Reasoning and Acting in Language Models. Preprint at arXiv. <https://doi.org/10.48550/arXiv.2210.03629>.
- [S19] Shinn, N., Cassano, F., Gopinath, A., Narasimhan, K., and Yao, S. (2023). Reflexion: language agents with verbal reinforcement learning. In Proceedings of the 37th International Conference on Neural Information Processing Systems (NeurIPS) (Curran Associates Inc.), pp. 8634–8652.
- [S20] Zhang, Y., Yang, J., Yuan, Y., and Yao, A.C.-C. (2023). Cumulative Reasoning with Large Language Models. Preprint at arXiv. <https://doi.org/10.48550/arxiv.2308.04371>.
- [S21] Kahneman, D. (2011). Thinking, Fast and Slow (macmillan).
- [S22] Harada, T., and Alba, E. (2021). Parallel Genetic Algorithms: A Useful Survey. *ACM Computing Surveys* 53, 1–39. <https://doi.org/10.1145/3400031>.
- [S23] Jun-ichi Kushida, Hara, A., Tetsuyuki Takahama, and Kido, A. (2013). Island-based differential evolution with varying subpopulation size. In 2013 IEEE 6th International Workshop on Computational Intelligence and Applications (IWCIA) (IEEE), pp. 119–124.

- [https://doi.org/10.1109/iwcia.2013.6624798.](https://doi.org/10.1109/iwcia.2013.6624798)
- [S24] Li, C., and Yang, S. (2008). An Island Based Hybrid Evolutionary Algorithm for Optimization. In Proceedings of the 7th International Conference on Simulated Evolution and Learning (Springer-Verlag), pp. 180–189. [https://doi.org/10.1007/978-3-540-89694-4\\_19](https://doi.org/10.1007/978-3-540-89694-4_19).