Enhancing LLMs for Power System Simulations: A Feedback-driven Multi-agent Framework

Response Letter

Mengshuo Jia[‡], Zeyu Cui[⋆], Gabriela Hug[‡]

[‡]*Power Systems Laboratory, ETH Zurich, 8092 Zurich, Switzerland*

[⋆]*Qwen Team at DAMO Academy, Alibaba Group, 311121 HangZhou, China*

In order to clearly distinguish the contents in this response letter, the comments of the editor and reviewers are provided in **blue**, the point-to-point responses are given in **black**, and the revised portions of the manuscript included here and in the revised paper are both in **red**. Besides, we use the following citing convention:

- Hyperlinks with dotted lines can direct to the corresponding responses by clicking, e.g., **Response to Comment 1.1**.
- References [1], [2], [3]...correspond to the references in the revised manuscript.
- References $[1]_r$, $[2]_r$, $[3]_r$...correspond to the references in this response letter.
- Fig. 1, 2, 3...correspond to the figures in the revised manuscript.
- Table I, II, III...correspond to the tables in the revised manuscript.

Response to the Editor

General Comment to the Authors: Two reviewers have showed major concerns about the sufficient implementation details, reproducibility clarity, performance comparison, and writing quality. Authors are suggested to add enough implementation details and conducted more comparisons so that the contributions in this paper can be improved. Note that fail to achieve the above requirements in next round may incur a rejection.

Response to the General Comment: We appreciate the editor's thoughtful comments and the opportunity to improve our manuscript. In response, we have made substantial revisions to address the concerns regarding implementation details, reproducibility clarity, performance comparison, and writing quality. Specifically, we have incorporated extensive enhancements to improve accessibility, rigor, and transparency.

First, to strengthen the **implementation details and enhance reproducibility clarity**, we have supplemented the following:

- The step-by-step process for constructing the triple-based document has been further detailed, along with an expanded discussion of the option document structure. The original documents are now publicly available in an open-source repository, with links prominently highlighted throughout the paper; see **Response to Comment 1.1**.
- The complete configuration settings for LLMs, RAG (including the embedding model), and the feedback loop have been explicitly added. API details for all evaluated schemes are now clearly presented in the version information of LLMs; see **Response to Comment 1.2**.
- Technical reviews, full datasets, and the parameters used in the simulation tasks have been supplemented to improve transparency and reproducibility; see **Response to Comment 1.4**.
- The complete pseudocode for both the embedding process and the full framework has been included, with step-by-step implementation details to facilitate reproducibility; see **Response to Comment 1.5**.
- The formal mathematical formulation of the scoring method has been rigorously specified to enhance reproducibility and ensure clarity in performance evaluations; see **Response to Comment 1.6**.
- The stopping criteria used in the framework have been further clarified through both textual descriptions and pseudocode, ensuring precise termination conditions; see **Response to Comment 1.8**.
- Further explanations have been provided for the evaluation metrics used in the cost analysis (e.g., input/output tokens, costs, and how they are computed); see **Response to Comment 1.12**.

Second, we have conducted **additional performance comparisons** to further validate the framework. In particular, we have reviewed the limited performance of non-LLM-based approaches first, and then included comparisons of our method against advanced LLM-based methods, specifically the supervised fine-tuning (SFT) for LLMs. Our experiments across diverse simulation environments and task complexities demonstrate that our framework significantly outperforms SFT, achieving accuracy improvements exceeding 40%. For details, see **Response to Comment 2.1**.

Furthermore, we have made **additional major revisions** to improve applicability, readability, and writing quality in general, as summarized below:

- The management of interdependencies and failure modes across modules has been further elaborated; see **Response to Comment 1.3**.

- The framework's capability to infer implicit terminologies from plain text has been explained in greater detail; see **Response to Comment 1.7**.
- The premise and underlying assumptions of the proposed framework have been further clarified to enhance understanding and applicability; see **Response to Comment 1.9**.
- A comprehensive summary of all implementation details has been given; see **Response to Comment 1.10**.
- A new future work section has been added, outlining potential solutions; see **Response to Comment 1.11**.
- The scalability of the cost analysis has been further discussed; see **Response to Comment 1.13**.
- The framework's adaptability to additional power system tools beyond DALINE and MATPOWER has been discussed in greater depth, demonstrating its flexibility for broader applications; see **Response to Comment 2.2**.
- Strategies for handling poorly structured error messages have been introduced; see **Response to Comment 2.3**.
- Sentence structures and terminology consistency have been refined throughout the manuscript to enhance readability; see **Response to Comment 2.4**.
- Citation credibility has been strengthened by increasing reliance on peer-reviewed sources, ensuring a well-supported research foundation; see **Response to Comment 2.5**.
- The academic writing style has been refined by removing first-person pronouns; see **Response to Comment 2.6**.

To sum up, we have carefully addressed all requirements raised by the reviewers, making extensive revisions to meet them. For further details, please refer to the point-by-point responses to the reviewers.

Response to Reviewer 1

General Comments to the Author: This paper introduces a conceptual framework to enhance large language models for power system simulations using a feedback-driven multi-agent approach, but it lacks sufficient implementation details, reproducibility clarity, and making it more of an exploratory idea than a practical, actionable contribution. The concerns are listed as follows.

Response to the General Comment: We appreciate the reviewer's time and effort in evaluating our work. The reviewer's thorough and insightful comments have been invaluable in enhancing the quality of our paper. In response, **we have incorporated extensive implementation details with formal and rigorous descriptions** to improve clarity, reproducibility, and practical applicability. Further clarifications have also been provided. Specifically, please refer to

- **Response to Comment 1.1** for a detailed explanation of the step-by-step construction of the option document and its open-source repository.
- **Response to Comment 1.2** for a comprehensive description of the configuration settings for LLMs, RAG (including how to integrate RAG), and the feedback loop.
- **Response to Comment 1.3** for an in-depth discussion on managing interdependencies and failure modes across modules.
- **Response to Comment 1.4** for the inclusion of technical reviews, complete datasets, and parameters for all the simulation tasks.
- **Response to Comment 1.5** for the full pseudocode of the embedding process and the complete framework implementation.
- **Response to Comment 1.6** for a formal mathematical formulation of the scoring method to enhance clarity and reproducibility.
- **Response to Comment 1.7** for a detailed explanation of how the framework infers implicit terminologies from plain text.
- **Response to Comment 1.8** for a refined and expanded description of the stopping criteria used in simulation tasks.
- **Response to Comment 1.9** for a clarification of the premise underlying the proposed framework.
- **Response to Comment 1.10** for a summary of all implementation details incorporated to improve transparency and reproducibility.
- **Response to Comment 1.11** for a newly added future work section, outlining potential solutions.
- **Response to Comment 1.12** for clarifications on model differences, evaluation fairness, and metric computation.
- **Response to Comment 1.13** for an in-depth discussion on the scalability of cost analysis and computational efficiency.

Below, we provide the revisions made in response to each comment.

Comment 1.1: The paper introduces a framework but omits the technical specifics required to reproduce the work. For example: No details are provided on the algorithms or techniques used to construct the triple-based structured option document.

Response to Comment 1.1*:* In the revised paper, we have detailed our construction approach for the document, and highlighted the link to share all the documents used for the evaluations. Specific revisions are given below.

First, we have added the details on our approach that generates the document in the revised paper.

Secion II. Enhanced RAG Module→ II-B Triple-based Structure Design for Knowledge Base

...

To overcome these issues, this paper proposes an additional, easy-to-construct retrieval repository: a triple-based structured option document. The following details the approach to build such a document for a given simulation tool:

- **Step 1:** ChatGPT-4o is employed to parse the simulation manual and automatically extract key option entities. This process produces a list of options directly from the manual, including option names, potential default values/formats, and descriptions.
- **Step 2:** The same LLM is further utilized to analyze the textual context in which the options appear, thereby determining the logical relationships between each extracted option and its associated simulation function.
- **Step 3:** The outputs from Steps 1 and 2 are organized into a structured text document using a triple format, where each triple comprises: (i) the option name with its default value/format, (ii) the corresponding function dependency (linking each option and its related functions), and (iii) the description of the option along with the choices of its value.
- **Step 4:** Finally, domain experts review and refine the automatically generated document to ensure that the logical associations between options and functions are accurate.

Second, it is important to note that the method described above is not mandatory; the entire process can also be performed manually, provided that readers understand the document's structure. To this end, we have included the option documents for both DALINE and MATPOWER in the supporting materials, available here. These materials not only facilitate reproducibility but also serve as templates for readers. Although the original manuscript also provided these materials, the material link was in the Case Study section. This placement may have caused the readers to overlook such materials. In the revised paper, we have prominently highlighted the template immediately after the description of the option document:

Secion II. Enhanced RAG Module → II-B Triple-based Structure Design for Knowledge Base

...

Note that the inclusion of the function dependency in the document enables retrieval of logical relationships. As demonstrated in case studies, this supplementary repository significantly enhances retrieval efficiency and improves the accuracy of simulation code generation by preserving the logical context. For detailed templates, please refer to the examples of retrieval documents provided [here].

Finally, we would like to clarify that constructing the triple-based structured option document does not require highly sophisticated algorithms. Our primary contribution lies in demonstrating the significant performance improvement achieved by using this document structure. Hence, the key focus is on the document structure itself, rather than the algorithm used to build it.

Comment 1.2: The exact configuration of the LLM and integration of the enhanced RAG module is not explained.

Response to Comment 1.2: In the revised paper, we have added the LLM and RAG configurations, highlighted the model versions of LLMs throughout the paper, expanded the explanation of RAG integration, and illustrated the RAG integration with pseudocode. Specific revisions are provided as follows.

First, the complete configuration settings used in the paper are now added to the revised paper:

Secion V. Case Study → V-A Settings

...

Meanwhile, the configurations used in all the evaluations are detailed in Table II.

TABLE II: Configuration settings for evaluations (for the LLM model versions, please see Table I or the captain of each evaluation figure/table; for all the prompt files, external documents, and simulation tasks, please refer to [here])

RAG Configuration		
Parameter	**Value**	**Description**
Chunk size for .txt	30	Number of words per chunk for text files
Chunk size for .pdf	50	Number of words per chunk for PDF files
Embedding Model	`v1`	See [here] for the model `text-embedding-v1`
Return number	20	Number of top-relevant chunks to return
LLM Configuration		
Parameter	**Value**	**Description**
Temperature	0.1	Randomness (lower = more predictable)
Max Tokens	4096	Max. length of the output in tokens
Top P	1	Probability threshold for token selection
Frequency Penalty	0	Zero means common words are not suppressed
Presence Penalty	0	Zero allows natural repetition when needed
Acting Configuration		
Parameter	**Value**	**Description**
DALINE $N^t_{\max}$	3	Max. attempt number for DALINE task t
MATPOWER $N^t_{\max}$	5	Max. attempt number for MATPOWER task t

Second, the information of LLM model versions is now clearly highlighted in the following locations throughout the revised paper:

- In the caption of Table I, which summarizes all the LLM models used in this paper for comparisons.
- In the caption of each evaluation figure or table, to facilitate result understanding and reproducibility clarity. For example:

Secion V. Case Study → V-C Evaluation on MATPOWER

...

- Fig. 9: Scores achieved by each evaluated scheme in individual attempts when handling complex tasks. The automatic error correction mechanism aids schemes in correcting their behavior when encountering errors in initial attempts (simulation environment: MATPOWER; GPT4o version: gpt-4o-2024-05-13; open-source repository for all tasks and results: [link]).
- Fig. 10: Success rates for each scheme, broken down by all tasks combined, complex tasks only, standard tasks only, as well as for the first attempt success rate and the final attempt success rate (simulation environment: MATPOWER; GPT4o version: gpt-4o-2024-05-13; `o1p-Sole` is only tested by complex tasks; open-source repository for all tasks and results: [link]).
- ...

Third, in addition to the details provided in Section III-C and Fig. 4 regarding RAG integration, we have expanded on this integration in the revised paper as follows:

Secion III. Enhanced Reasoning Module → III-C Knowledge Integration

...

The above reasoning actions heavily rely on information drawn from both the simulation request and supplementary knowledge, comprising:

- *Static Basic Knowledge*: Supplies the agent with foundational information on essential functions and syntax rules pertinent to the simulation tool. This static knowledge serves as a base reference and reminder for the agent to consult when generating code. Note that such knowledge is tool-dependent.
- *Dynamic Retrieval Knowledge*: Complements static knowledge by incorporating real-time, request-specific details. This is achieved through the integration of the aforementioned RAG module, which retrieves relevant information on option formats, values, and function dependencies, for accurate code generation.

The integration of the RAG module into the LLM-based coding agent (from the reasoning module) is established through placeholders in the prompt. These placeholders serve as symbolic flags that are dynamically replaced by retrieval results before the prompt is sent to the coding agent. As shown in the "Retrieval Knowledge" section of Fig. 4 (b), during runtime, the enhanced RAG module retrieves simulation-specific information and substitutes these placeholders with up-to-date details on simulation functions, options, and parameters. Consequently, the coding agent generates simulation code based on both this dynamically retrieved information and the static knowledge embedded in the prompt.

Fourth, we have added pseudocode for the full framework, which further details the integration of RAG into the feedback loop. For the pseudocode, please refer to **Response to Comment 1.5**.

Comment 1.3: The proposed LLM structure, while innovative in concept, suffers from a fundamental flaw: it overly relies on modular enhancements (RAG, reasoning modules, feedback loops) without addressing their interdependencies or failure modes.

Response to Comment 1.3*:* We agree with the reviewer that our study demonstrates significant progress through modular enhancement. However, our approach also integrates an interactive feedback mechanism to manage

interdependencies between modules and address their failure modes. The clarifications are provided below, followed by the specific revision made to the manuscript.

First, to manage interdependencies, our approach employs an environmental acting module with error feedback, as detailed in Section IV. This module is not an isolated add-on. It continuously monitors simulation executions and generates error reports when necessary. These reports trigger adaptive adjustments in both the RAG and reasoning modules. The outputs of these modules are then fed back into the environmental acting module. This continuous loop ensures that all interdependencies are effectively managed.

Second, our framework actively anticipates and addresses potential failure modes rather than overlooking them. The error-feedback mechanism is designed to detect failures in both the RAG and reasoning modules. A failure in either module can lead to simulation errors. Once a failure is detected, the error-feedback mechanism initiates corrective actions iteratively using adaptive prompts, which trigger updated retrievals in the RAG module and adjustments in the reasoning module. This process results in a dynamic management of failures. Our experimental results—particularly in Fig. 9 and Fig. 12—confirm that failures are corrected after unsuccessful attempts.

In the revised paper, we have clearly highlighted that interdependencies are managed through our adaptive feedback loop and that failure modes are dynamically addressed:

> **Secion IV. Environmental Acting Module with Feedback → IV-C Enhanced RAG and Reasoning Module Interplay**
>
> ...
>
> The error report and feedback are then processed as a new request by the retrieval agent in the enhanced RAG module. This agent retrieves relevant information based on the error report (the query planning can also be applied to error reporting by simply replacing the identification of function/option keywords with the identification of error-related keywords). The retrieved information is then passed to the enhanced reasoning module. The coding agent there uses both the retrieval results and the correction request to revise the simulation code. The updated code is fed back into the environmental acting module. This loop continues until the code meets all requirements or the stopping criterion is reached.
>
> This design achieves two main objectives. First, the system continuously monitors simulation executions and triggers adaptive adjustments as needed, ensuring that all module interdependencies are effectively managed. Second, it handles failure modes. The error-feedback mechanism detects failures in both the RAG and reasoning modules—failures that can lead to simulation errors. Once a failure is detected, the mechanism initiates iterative corrective actions using adaptive prompts, which trigger updated retrievals and adjustments, thereby dynamically managing module failures. This conclusion is further reinforced by the case studies in Section V.

Comment 1.4: The framework is tested on DALINE and MATPOWER, but the datasets or input parameters for the simulation tasks are not presented. I see there are some links shared to point to, but not sufficient enough to have a good insight for the paper results.

Response to Comment 1.4*:* To provide sufficient information on the datasets/parameters for simulations, in the revised paper, we have reviewed all simulation tasks, provided the open-source repository for the dataset of tasks, detailed the scoring method and stopping criteria for simulations, and included the parameters/versions of LLMs. The specific revisions are as follows.

First, to provide readers with a comprehensive understanding, in the revised paper we systematically reviewed all simulation tasks for both DALINE and MATPOWER:

Secion V. Case Study → V-A Settings

...

Thirdly, 34 simulation tasks were used to evaluate the proposed framework on DALINE. Similarly, for MATPOWER, 35 simulation tasks have been defined, including 8 complex tasks and 27 standard tasks.

- The simulation tasks of DALINE span from basic data generation to advanced workflows that integrate data creation, data corruption, noise handling, outlier filtering, model training, method comparison, result visualization, and full-cycle management. The simulation suite exercises all distinct functions in DALINE across various power system cases. The tasks employ multiple modeling methods (e.g., Least Squares with Huber Weighting Function, Partial Least Squares with Clustering, Ridge Regression with Voltage-angle Coupling, Locally Weighted Ridge Regression, State-independent Voltage-angle Decoupled Method with Data Correction, to name a few) and a wide array of parameters (sample sizes, base types, noise levels, outlier techniques, cross-validation settings, method hyperparameters, etc.).
- The simulation tasks of MATPOWER span standard AC and DC optimal power flow (or just load flow) analyses as well as advanced continuation power flow (CPF) studies. They employ a variety of solvers and algorithms—including Newton-Raphson, Fast-Decoupled (both XB and BX versions), Gauss-Seidel, Implicit Z-bus Gauss, MIPS, fmincon, and GUROBI—across numerous test cases such as the IEEE 9-, 14-, 30-, 57-, 118-, and 145-bus systems, along with specialized cases like the 4-bus, 6-bus, 39-bus New England, 51-bus radial, 60-bus Nordic, and IEEE RTS 24-bus systems. Key parameters such as maximum iterations, mismatch tolerances, coordinate representations, and branch flow constraints were systematically varied. In the CPF analyses, both natural and pseudo arc length parameterizations were explored with adaptive step sizing, explicit nose point detection, and generation/load scaling. Additional configurations addressed generator reactive power limits, voltage setpoints, alternative formulations (current versus power balance), and solver-specific settings like gradient, optimality, and termination tolerances.

Overall, these tasks comprehensively cover both basic and advanced functionalities of DALINE and MATPOWER across a broad range of power system analysis scenarios. A selection of representative tasks is given in Fig. 6, intended as an exemplary overview; for all simulation tasks, refer to [here].

Second, we have included the full set of simulation tasks for both DALINE and MATPOWER. This set covers all simulation tasks, the benchmark code, and the code generated during each attempt for every scheme, totaling 870

coding files. Due to the large volume of these files, they are available via an open-source repository (see [here]). To ensure visibility, reminders with the repository link are further highlighted throughout the revised paper, especially in the caption of each evaluation figure and table. For example:

Secion V. Case Study → V-B Evaluation on DALINE

...

- Fig. 7: Distribution of scores achieved across attempts for each evaluated scheme (simulation environment: DALINE; GPT4o version: gpt-4o-2024-05-13; open-source repository for all tasks and results: [link]).
- Fig. 8: Success rates for each scheme, itemized by all tasks combined, complex tasks only, standard tasks only, as well as for the first attempt success rate and the final attempt success rate (simulation environment: DALINE; GPT4o version: gpt-4o-2024-05-13; open-source repository for all tasks and results: [link]).
- ...

Third, we have supplemented the following content to better present the parameters or implementation details: (i) we have detailed the scoring methodology for simulation tasks, as given in **Response to Comment 1.6**. (ii) we have further explained the stopping criteria used for simulation tasks, as provided in **Response to Comment 1.8**. (iii) We now provide detailed configuration parameters for the evaluations, which are presented in **Response to Comment 1.2**.

Finally, please also note that the revised manuscript still retains the following details from the original manuscript:

- Representative examples of simulation tasks for DALINE and MATPOWER in Fig. 6.
- Version information for DALINE and MATPOWER in Section V-A.
- Parameters for attempts described in Section V-A.
- The scoring methodology outlined in Section V-A.

With all the above information, we believe that the reader has sufficient information to understand and reproduce our simulation results.

Comment 1.5: Core methods like the feedback-driven mechanism, query planning strategy, and reasoning enhancements are only described conceptually, with no pseudocode or technical examples.

Response to Comment 1.5: In the revised paper, we have added the complete pseudocode for (i) the embedding process and (ii) the proposed framework. The pseudocode uses self-explanatory pseudo-functions to illustrate the internal processes within the core modules and the interplay among them. Details are as follows:

Secion II. Enhanced RAG Module

...

Algorithm 1 details the configuration settings and process for the embedding used in this paper.

Algorithm 1: Embedding for External Documents

Input:
datasetPath: Paths to external documents
chunkSize: Words per chunk
embeddingModel: Chosen text embedding model
apiKey: Credential for embedding
Output:
vectorDB: Vector database for documents

```
// Step 1: Chunk;
chunks ← splitIntoChunks(datasetPath, chunkSize)
// Step 2: Embedding;
vectors ← embedChunks(chunks, embeddingModel, apiKey)
// Step 3: Store in Vector DB;
vectorDB ← storeVectors(vectors, apiKey)
```

Secion IV. Environmental Acting Module with Feedback → IV-E Pseudocode for the Full Framework

...

Algorithm 2 presents the complete pseudocode for the proposed framework. The algorithm employs self-explanatory pseudo-functions and pseudo-options to illustrate the core modules, including their internal processes and the interplay among them.

Algorithm 2: Feedback-driven Multi-agent Framework

Input: *modelVer*: Model version from Table I
config: Configuration settings from Table II
ragPrompt: Query planning prompt from Fig. 2
reasonPrompt: Structured reasoning prompt from Fig. 4
errTemplate: Error report template from Fig. 5
vectorDB: Vector database from Algorithm 1
task: Simulation task
maxAttempts: Max attempt number
apiKey: Credentials for LLMs
returnNum: Number of retrievals to return
Output: *simResult*: Simulation result
code: Generated code

```
// Preparation;
attemptTime ← 0;
chatHistory ← emptySet;
env ← activateSimulationEnvironment()
// Enhanced RAG Module;
retrievalLLM ← setupLLM(config, modelVer, ragPrompt, apiKey);
keywords ← generateKeywords(retrievalLLM, task);
retrievalInfo ← parallelRetrieval(retrievalLLM, keywords, vectorDB, returnNum);
// Enhanced Reasoning Module;
reasonPrompt ← insertRetrieval(reasonPrompt, retrievalInfo);
codeLLM ← setupLLM(config, modelVer, reasonPrompt, apiKey);
[code, chatHistory] ← generateCode(codeLLM, task, chatHistory);
// Environmental Acting Module with Feedback;
[simResult, err] ← runSimulation(code, env);
while err ≠ null and attemptTime ≤ maxAttempts do
    errReport ← generateErrorReport(code, err, errTemplate);
    keywords ← generateKeywords(retrievalLLM, errReport);
    newRetrievalInfo ← parallelRetrieval(retrievalLLM, keywords, vectorDB, returnNum);
    reasonPrompt ← insertRetrieval(reasonPrompt, newRetrievalInfo);
    codeLLM ← setupLLM(config, modelVer, reasonPrompt, apiKey);
    [code, chatHistory] ← generateCode(codeLLM, errReport, chatHistory);
    [simResult, err] ← runSimulation(code, env);
    attemptTime ← attemptTime + 1;
if err ≠ null then
    reportFailure(code);
    saveFailedCode(code);
else
    outputResult(simResult);
    saveSimulationCode(code);
```

Comment 1.6: Success rates (e.g., 93.13% on DALINE and 96.85% on MATPOWER) are highlighted, but the scoring process and metrics are inadequately explained.

Response to Comment 1.6: To provide a more adequate and rigorous explanation, the revised paper now presents the scoring methodology and metric definitions using formal mathematical formulas:

Secion V. Case Study → V-A Settings

...

Finally, to evaluate each scheme's performance, we compute a success rate based on the points obtained across multiple simulation tasks. Let T denote the total number of simulation tasks. For each task t ($1 \leq t \leq T$), let $N_{\max}^{(t)}$ be the maximum number of allowed attempts for that task and let $P_{t,i}$ be the points awarded on attempt i (with $1 \leq i \leq N_{\max}^{(t)}$). The scoring for task t at attempt i is defined as follows:

- $P_{t,i} = 100$ points if the simulation result is exactly correct with no irrelevant settings in the code.
- $P_{t,i} = 50$ points if the simulation result is correct but contains irrelevant settings.
- $P_{t,i} = 0$ points if the simulation result is incorrect.

Subsequent attempts are made only if the previous attempt resulted in an error, and any unused attempts are assigned the same score as the last executed attempt. Thus, the total score for task t is given by:

$$S_t = \sum_{i=1}^{N_{\max}^{(t)}} P_{t,i},$$

with a maximum possible score of $100 \times N_{\max}^{(t)}$ for that task. Finally, the overall success rate across all tasks is computed as:

$$R_{\text{overall}} = \frac{\sum_{t=1}^{T} S_t}{\sum_{t=1}^{T} \left(100 \times N_{\max}^{(t)}\right)} \times 100\%.$$

In the following experiments, $N_{\max}^{t} = 3$ ($\forall t$) is set for DALINE tasks, and $N_{\max}^{t} = 5$ ($\forall t$) is set for MATPOWER tasks, reflecting the increased complexity of the latter.

Comment 1.7: The adaptive query planning strategy splits requests into function-related and option-related sub-queries, which improves retrieval precision. However, it is unclear how this approach handles ambiguous or incomplete simulation requests, where explicit function or option specifications are missing.

Response to Comment 1.7: We agree with the reviewer that when simulation requests lack specificity and terminology, it presents challenges for natural language coding.

However, we clarify that our query planning strategy can infer implicit terminologies from plain text. Please refer below to the detailed explanation in the revised paper, which includes a practical example:

Secion II. Enhanced RAG Module → II-A Adaptive Query Planning

...

Remark: The adaptive query planning strategy goes well beyond simple keyword extraction. It employs semantic recognition via few-shot CoT, which can translate descriptions into relevant keywords. This enables the retrieval agent to infer implicit simulation functions and options from context—even when explicit terms are missing. For example, consider the request: "*For the IEEE 24-bus Reliability Test System, solve the AC optimal power flow (OPF) problem with the de-commitment of expensive generators*," which is the initial request in complex task 7 for MATPOWER (see Section V for more details). Rather than extracting generic phrases like "AC optimal power flow" or "de-commitment," the retrieval agent—guided

by few-shot CoT prompting for function mapping—directly identifies “runuopf” as the keyword, i.e., the specific MATPOWER function for OPF with de-commitment.

Second, when simulation requests are inherently ambiguous or underspecified, it is challenging for any automated method—not just ours—to accurately determine the user’s intent. This ambiguity arises from the initial lack of specificity in the user’s input, rather than a limitation of natural language coding techniques. Natural language coding cannot serve as a “mind-reader” to deduce the complete thought process when specific needs are not clearly expressed.

Nonetheless, this challenge might be addressed through a “human-in-the-loop” design. One promising future direction is to incorporate an interactive dialogue stage before the main framework is activated. In this stage, an agent automatically assesses the specificity of the initial request and, if necessary, triggers targeted clarifying questions to help users refine their requirements. This iterative process ultimately leads to more precise retrieval and improved code generation. In the revised paper, we outline this study as an additional future research direction:

Secion VI. Conclusion

...

Second, when simulation requests are inherently ambiguous or underspecified, accurately determining the user’s intent becomes difficult for any automated method—not just the framework presented in this paper. This ambiguity primarily arises from the initial lack of detail in the user’s input, rather than any intrinsic limitation of natural language coding. To address this, a promising future direction is to integrate an interactive dialogue stage into the framework. In this approach, an autonomous agent would evaluate the specificity of the initial request and, if necessary, prompt the user with targeted clarifying questions. This human-in-the-loop strategy could help refine the requirements, leading to more precise retrieval and improved code generation.

Comment 1.8: The stopping criterion is briefly mentioned but not quantified. It is unclear how the system decides whether further attempts at correction are worthwhile. Also, the stopping criteria for the feedback loop in the environmental acting module are not clearly defined. It vaguely states ”if the stopping criterion is met, the process is terminated,” without specifying what conditions fulfill this criterion.

Response to Comment 1.8*:* In the revised paper, we have incorporated pseudocode for the proposed framework, specifically Algorithm 2, which formally defines the stopping criteria. Please refer to **Response to Comment 1.5** for further details.

Here, we would like to further clarify: our framework actually defines a complete attempt as a full cycle—from retrieval through reasoning to execution. At the end of a cycle, if an error persists at the end of an attempt and the maximum number of attempts has not been reached, i.e.,

$$\texttt{while (error} \neq \texttt{null and attemptTime} \leq \texttt{maxAttempts)}$$

the system automatically initiates another cycle (i.e., a new attempt). The loop terminates when either no error is detected or the maximum number of attempts is reached. In the former case, the simulation result is returned, whereas

in the latter, a failure is reported. Consequently, the stopping criteria for feedback and attempts are fundamentally equivalent—both are governed by the error detection and the attempt count, as given above.

This stopping approach has demonstrated high success rates, as shown in our case studies. However, it does not guarantee a 100% success rate, because certain ”non-execution-bug” failures may go undetected. This limitation is not unique to our framework, but reflects the inherent challenge of fully capturing subtle errors in simulation outputs. To address this, we aim to develop an enhanced error-detection mechanism, as mentioned in the future work section of the original manuscript. Now, this future work section has been expanded with potential solutions, as detailed below:

Secion VI. Conclusion

...

However, several critical future challenges still remain. **First**, while the proposed framework demonstrates significant improvements over existing schemes and supervised fine-tuning, achieving a perfect 100% success rate for LLM-based simulations remains an open challenge. For instance, the proposed framework sometimes fails to detect “non-execution-bug” failures, resulting in unaware inaccurate outputs that cannot be automatically corrected. A potential avenue for future work is to explore the use of a RAG-supported, fine-tuned LLM-based code-checking agent, which would verify executed simulation code and identify hidden errors that the error-reporting mechanism may overlook.

Comment 1.9: The paper mentions that the framework achieves high success rates, but it lacks analysis on how the system performs when new, unseen tools are introduced that lack detailed manuals or structured documentation.

Response to Comment 1.9*:* We would like to clarify that our framework is designed for scenarios where simulation tools are within the realm of human learnability. In other words, if a human can learn and effectively use an unseen tool with its available documentation, then our framework—through its integrated RAG, reasoning, and feedback-based acting modules—can replicate this process and further enhance efficiency. This principle is the core objective of our work: to accelerate the effective use of simulation tools based on the premise that these tools are learnable with sufficient documentation, whether or not the tools were seen before by LLMs.

Conversely, expecting an LLM to directly operate with a unseen tool that lacks detailed documentation, amounts to asking for the impossible. If even a human would struggle with such a tool, it is also unreasonable to expect an LLM to succeed without adequate guidance. One theoretical solution would be to supply the entire underlying code to the LLM each time it is called, thereby compensating for missing documentation. However, this approach would be prohibitively expensive and complex due to the sheer volume and fragmentation of the code. An alternative strategy could be to provide the complete underlying code of a simulation tool to the LLM once, allowing it to generate a readable, structured documentation—such as a manual detailing all functions and options. This generated documentation could then be integrated into our framework, making our approach still applicable.

In the revised paper, we have made the following clarification to emphasize the condition of using our framework:

Secion IV. Environmental Acting Module with Feedback → IV-E Pseudocode for the Full Framework

...

Remark: It is important to emphasize that the proposed framework is built on the premise that simulation tools are learnable by humans through sufficient documentation. If a human can use a simulation tool based on its manuals, then the proposed integrated RAG, reasoning, and feedback modules can replicate and enhance this process to improve efficiency. Conversely, expecting an LLM to directly operate with a unseen tool, which lacks detailed documentation, is unrealistic, as even human users would struggle in such cases. Hence, for scenarios where documentation is absent, it is suggested to generate necessary documents, possibly with LLMs' help. These documents can then be incorporated into the proposed framework.

Comment 1.10: The architecture and workflow diagrams are high-level but lack specific implementation details, such as APIs used, data structures for triple-based documents, or query embedding models.

Response to Comment 1.10: In the revised paper, we have added the following content to provide specific implementation details, including APIs, data structures for triple-based documents, embedding models, and more:

- The step-by-step process for constructing the triple-based document has been further detailed, along with an expanded explanation of the option document structure. The original documents are now available in an open-source repository, with the link highlighted throughout the paper; see **Response to Comment 1.1** for details.
- The complete configuration settings for LLMs, RAG (including the embedding model), and the feedback loop have been added. API details for different evaluated schemes are highlighted in the version information of LLMs throughout the paper; see **Response to Comment 1.2**.
- Technical reviews, full datasets, and the parameters used in the simulation tasks have been supplemented; see **Response to Comment 1.4**.
- The complete pseudocode for both the embedding process and the full framework has been included, with step-by-step implementation details; see **Response to Comment 1.5**.
- The formal mathematical formulation of the scoring method has been rigorously specified to enhance reproducibility; see **Response to Comment 1.6**.
- The stopping criteria used in the framework have been further clarified through both textual descriptions and pseudocode; see **Response to Comment 1.8**.
- The metrics used for cost analysis have been further explained in **Response to Comment 1.12**.

With the above additions, we aim to enhance the paper's reproducibility, clarity, and accessibility for readers.

Comment 1.11: The conclusion highlights error-detection as a future area of improvement, but no preliminary results or methodologies are proposed. Synchronizing multiple tools is mentioned as a challenge but is not demonstrated.

Response to Comment 1.11: We have revised the future work section in the Conclusion, focusing on future directions where we have developed potential solutions or meaningful ideas:

Secion VI. Conclusion

...

However, several critical future challenges still remain. **First**, while the proposed framework demonstrates significant improvements over existing schemes and supervised fine-tuning, achieving a perfect 100% success rate for LLM-based simulations remains an open challenge. For instance, the proposed framework sometimes fails to detect "non-execution-bug" failures, resulting in unaware inaccurate outputs that cannot be automatically corrected. A potential avenue for future work is to explore the use of a RAG-supported, fine-tuned LLM-based code-checking agent, which would verify executed simulation code and identify hidden errors that the error-reporting mechanism may overlook. **Second**, when simulation requests are inherently ambiguous or underspecified, accurately determining the user's intent becomes difficult for any automated method—not just the framework presented in this paper. This ambiguity primarily arises from the initial lack of detail in the user's input, rather than any intrinsic limitation of natural language coding. To address this, a promising future direction is to integrate an interactive dialogue stage into the framework. In this approach, an autonomous agent would evaluate the specificity of the initial request and, if necessary, prompt the user with targeted clarifying questions. This human-in-the-loop strategy could help refine the requirements, leading to more precise retrieval and improved code generation.

Comment 1.12: It is not clear how each ChatGPT model is different that the other to have the fair comparison and how you computed the metrics mentioned for fair evaluation. like pricing, tokens, you shall really elaborate on every detail to make readers familiar with all technical terms mentioned for this work.

Response to Comment 1.12: In the following response, we first clarify the differences among the evaluated LLM models, then describe our approach to ensuring a fair evaluation, and finally detail the metrics used (e.g., tokens and pricing) along with an explanation of these technical terms. For further elaboration on additional technical details—including methodology, documentations, evaluation strategies, configurations, and models, please refer to **Response to Comment 1.10**.

First, for model differences, the extended Table I now clearly outlines the differences among the evaluated schemes, which vary in both the LLM models and the techniques applied. Most schemes are accessed via API. However, to simulate the ChatGPT website experience, we also evaluated `ChatGPT4o` and `o1-preview`—leveraging their full capabilities (e.g., document uploading for RAG). Our goal is to assess how far one can go using only the website service provided by OpenAI. There are two noteworthy points:

- `ChatGPT4o` typically employs a more advanced model than the API version. For example, the API model is either `gpt-4o-2024-05-13` or `gpt-4o-2024-08-06`, whereas the website version runs on `chatgpt-4o-latest`.
- We use two API versions—`gpt-4o-2024-05-13` and `gpt-4o-2024-08-06`—to ensure a fair comparison. The newer version is required because our additional comparison involves supervised fine-tuning for LLMs, and OpenAI permits fine-tuning only on `gpt-4o-2024-08-06`. Accordingly, we re-evaluate certain schemes with `gpt-4o-2024-08-06`. For transparency, version information is provided after each evaluation.

Second, to futher ensure a fair evaluation, we strictly follow the "one-variable-at-a-time" approach using identical testing simulation tasks for all schemes:

TABLE I: Evaluated schemes by distinct combinations of the proposed strategies within the framework (GPT4o: APIs of gpt-4o-2024-05-13 and gpt-4o-2024-08-06, with the exact version specified in each evaluation; CGPT4o: ChatGPT4o; o1p: o1-preview)

	GPT4o Full	GPT4o PR	GPT4o RSR	GPT4o SR	GPT4o Sole	GPT4o NC	GPT4o NP	GPT4o NS	GPT4o NR	GPT4o NCS	GPT4o RSRNW	CGPT4o R	o1p Sole	GPT4o Sole-SFT
Query Planning	✓	✓				✓	✓	✓		✓	✓			
Triple-based Structured Option Document	✓	✓	✓	✓		✓		✓		✓	✓	✓		
Chain of Thought Prompting	✓		✓				✓	✓	✓		✓			
Few-Shot Prompting	✓		✓			✓	✓		✓		✓			
Static Basic Knowledge	✓		✓			✓	✓	✓	✓	✓	✓			
Environmental Acting and Feedback	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Proposed RAG	✓	✓				✓	✓	✓		✓	✓			
Standard RAG			✓	✓										
OpenAI's Built-in RAG												✓		
OpenAI's Built-in Supervised Fine-tuning														✓
Well-developed Error-reporting System	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓

- **Single Technique Comparison:** When focusing on one technique, we compare schemes that use the same LLM model and technique set, differing only in the technique under study. For example, we compare `GPT4o-Full` with `GPT4o-NC` (both using `gpt-4o-2024-05-13`) to evaluate the impact of chain-of-thought.
- **Single Module Comparison:** When isolating a module, we compare schemes with the same LLM model and module set except for the module being evaluated. For instance, we compare `GPT4o-Full` with `GPT4o-PR` (both using `gpt-4o-2024-05-13`) to assess the effect of the enhanced reasoning module.
- **Overall Approach Comparison:** We compare our full framework as a complete method with the ChatGPT4o website version (which includes RAG capability), even though `ChatGPT4o` uses the more advanced `chatgpt-4o-latest` model.

By following the above rules and disclosing the model versions throughout the paper, we ensure transparency and fairness in our evaluation process.

Third, in the revised paper, we have clarified the technical terms "token" and "expense," and explain their meaning and computation right after Table IV:

Secion V. Case Study → V-E Cost Analysis

...

TABLE IV: Average cost analysis of `GPT4o-Full` per task*

Environment	Time (sec.)	Input Token†	Output Token	Expense (USD) ‡
DALINE	29.446	7882.294	168.353	0.014
MATPOWER	32.703	5338.514	274.371	0.013

† A token is a fundamental unit of text, typically representing fragments of words. Tokens quantify both the input (text submitted to the LLM via API) and the output (text or code generated by the LLM via API).
‡ Expense refers to the monetary cost incurred for processing these tokens, as charged by the API provider (OpenAI in this study).
* The API provider automatically computes the total tokens and associated expenses. The average values here are calculated by dividing the total tokens (and corresponding expenses) used across all simulation tasks with `GPT4o-Full` by the number of tasks.

Comment 1.13: While token costs and task execution times are mentioned, there is no discussion of how the system scales with increased task complexity.

Response to Comment 1.13*:* In this response, we first clarify the purpose of the cost analysis. Then, we explain that the overall execution time and token cost do not necessarily increase with task complexity. Finally, we provide the revision made to the paper.

First, regarding the purpose of the analysis, as acknowledged in the original manuscript, the reported execution time and token cost serve only as approximate indicators, given the inherent variability introduced by factors such as retrieval, reasoning, code execution, result aggregation, and iterative error correction. Hence, this cost analysis only aims to provide readers with a general sense of the efficiency of automated simulations within the scope of the specific simulation tasks used in this paper, rather than to establish precise quantitative relationships.

Second, the indicated total execution time aggregates several components (e.g., code generation and code execution). Although more complex simulation requests may yield longer input/output texts, the simulation environment's processing time remains largely independent once the code is generated. Therefore, the overall time does not necessarily scale with task complexity.

Third, while we use the length of task descriptions as a practical means to differentiate between standard and complex tasks, we acknowledge that task complexity lacks a universally rigorous definition. Consequently, there is no strict correlation between task complexity and the length of the request text, nor between task complexity and token cost. Additionally, iterative feedback loops and parallel processing introduce non-linear scalability effects, further complicating any direct relationship associated with scalability analysis.

In the revised paper, we have added the following clarifications to ensure a more precise interpretation of our cost analysis:

Secion V. Case Study → V-E Cost Analysis

...

Even in the current state, without any specific acceleration strategies, `GPT4o-Full` can execute approximately 120 simulation tasks per hour in DALINE and MATPOWER, with a total token cost of around 1.68 USD. Given this high efficiency and cost-effectiveness, the proposed framework presents a promising pathway to improve researcher productivity.

It must be emphasized, however, that the reported execution time and token cost serve only as approximate indicators, given the inherent variability introduced by factors such as retrieval, reasoning, code execution, result aggregation, and iterative error correction. This analysis aims to provide readers with a general sense of the efficiency of automated simulations within the scope of the specific simulation tasks, rather than to establish precise quantitative relationships.

Secion VI. Conclusion

...

(iv) The proposed framework enables LLMs to execute tasks efficiently, with each task completed in approximately 30 seconds at a token cost of only 0.014 USD (for the simulation tasks used in this paper)...

Response to Reviewer 2

General Comments to the Author: This paper proposes a feedback-driven, multi-agent framework to enhance the capabilities of large language models (LLMs) for power system simulations. The framework integrates an enhanced retrieval-augmented generation (RAG) module, an advanced reasoning module, and an environmental acting module with an error-feedback mechanism. Several suggestions are provided for the authors.

Response to the General Comment: We would like to thank the reviewer for the insightful suggestions. The reviewer's comments have been instrumental in refining our work, providing critical perspectives that significantly enhance the manuscript. In response, **we have incorporated new comparisons, discussions, and clarifications** to better demonstrate the framework's superiority, adaptability, and credibility. Specifically, please refer to:

- **Response to Comment 2.1** for a comparative evaluation of supervised fine-tuning (SFT) for LLMs in DALINE and MATPOWER, highlighting its limitations and showcasing the advantages of the proposed framework.
- **Response to Comment 2.2** for an extended discussion on the framework's adaptability to additional power system tools beyond DALINE and MATPOWER.
- **Response to Comment 2.3** for strategies to handle ambiguous or poorly structured error messages, improving robustness.
- **Response to Comment 2.4** for refinements in sentence structure and terminology consistency throughout the manuscript.
- **Response to Comment 2.5** for strengthening citation credibility by increasing reliance on peer-reviewed sources.
- **Response to Comment 2.6** for improvements in academic writing style, including the removal of first-person pronouns to maintain a formal tone.

The point-by-point responses are given below.

Comment 2.1: Incorporate comparisons with advanced methodologies, such as fine-tuning or hybrid frameworks, to highlight the framework's superiority, e.g., compare to the representative non-LLM-based approaches.

Response to Comment 2.1: The reviewer highlighted two major categories for comparison: LLM-based methods (e.g., fine-tuning) and non-LLM-based methods.

First, we would like to clarify that, traditional non-LLM approaches have achieved limited success in natural-language-to-code (NL-to-code) translation due to inherent constraints. Specifically, SWIM, proposed in $[1]_r$, synthesizes code by searching for relevant snippets and summarizing the results. However, its heavy reliance on code retrieval limits its ability to generalize beyond the search query. Similarly, the study in $[2]_r$ employed recurrent neural networks (RNNs) in a sequence-to-sequence setup, but this approach was restricted to generating fixed patterns of code and failed to produce coherent or complete logic. Moreover, JuICe, introduced in $[3]_r$, focused on NL-to-code tasks but was constrained to Python and typically generated code sequences averaging only 38 tokens—insufficient for practical applications. CodeGen, developed in $[4]_r$, marked an early transition toward large models for NL-to-code tasks and gained significant attention. While CodeGen utilized a transformer-based architecture

trained on a relatively modest dataset, its performance was limited by the computational and data constraints of its time, falling short of the capabilities of modern LLMs. Overall, these early methods were fundamentally constrained by the available technology and computational resources. In contrast, LLM-based approaches, trained on large-scale code–natural language corpora, inherently capture diverse syntactic, semantic, and structural patterns, significantly enhancing code understanding and generation. Therefore, comparing LLM-based approaches with non-LLM methods may create an unfair evaluation environment.

Given the evident superiority of LLM-based methods, and following the reviewer's suggestion, **we focus our comparison on advanced LLM methodologies: LLMs with supervised fine-tuning (SFT)**. Our experiments across diverse simulation environments and task complexities demonstrate that our proposed framework significantly outperforms SFT, achieving accuracy improvements exceeding 40%. We attribute this performance gap to (1) the inherent lossy compression process of SFT and (2) the limited parameter set adjusted during fine-tuning.

The following details the newly added comparison, as well as the revised Abstract and Conclusion sections:

TABLE I: Evaluated schemes by distinct combinations of the proposed strategies within the framework (GPT4o: APIs of gpt-4o-2024-05-13 and gpt-4o-2024-08-06, with the exact version specified in each evaluation; CGPT4o: ChatGPT4o; o1p: o1-preview)

	GPT4o Full	GPT4o PR	GPT4o RSR	GPT4o SR	GPT4o Sole	GPT4o NC	GPT4o NP	GPT4o NS	GPT4o NR	GPT4o NCS	GPT4o RSRNW	CGPT4o R	o1p Sole	GPT4o Sole-SFT
Query Planning	✓	✓				✓	✓	✓		✓	✓			
Triple-based Structured Option Document	✓	✓	✓	✓		✓		✓		✓	✓	✓		
Chain of Thought Prompting	✓		✓				✓	✓	✓		✓			
Few-Shot Prompting	✓		✓			✓	✓		✓		✓			
Static Basic Knowledge	✓		✓			✓	✓	✓	✓	✓	✓			
Environmental Acting and Feedback	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Proposed RAG	✓	✓				✓	✓	✓		✓	✓			
Standard RAG			✓	✓										
OpenAI's Built-in RAG												✓		
OpenAI's Built-in Supervised Fine-tuning														✓
Well-developed Error-reporting System	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓

Section V. Case Study → V-D. Comparison with Supervised Fine-tuning

To further verify the proposed framework's performance, it was compared with supervised fine-tuning (SFT)[1], a well-established approach known to enhance LLM performance on specific tasks—including unseen tasks such as translation and natural language inference [27]. In order to precisely demonstrate the improvements introduced by SFT, `GPT4o-Sole` has been employed as the foundation for SFT training[2], denoted by `GPT4o-Sole-SFT` in Table I.

The initial comparison was conducted on MATPOWER tasks. For this evaluation, 50 random tasks (following the recommendation from OpenAI) were generated with preferred coding responses, producing a supervised training dataset of 24,249 tokens (the dataset is available [here]). The model was fine-tuned for 3 epochs, with a batch size of 1 and a learning rate multiplier of 2 (all automatically configured). Fig. 12 illustrates the performance of `GPT4o-Full`, `GPT4o-Sole`, and `GPT4o-Sole-SFT`,

[1]The OpenAI's SFT approach was used in this paper; see [here] for details.

[2]Since gpt-4o-2024-08-06 is the only version available for SFT, all GPT4o-related schemes in Section-VD use this version for consistency.

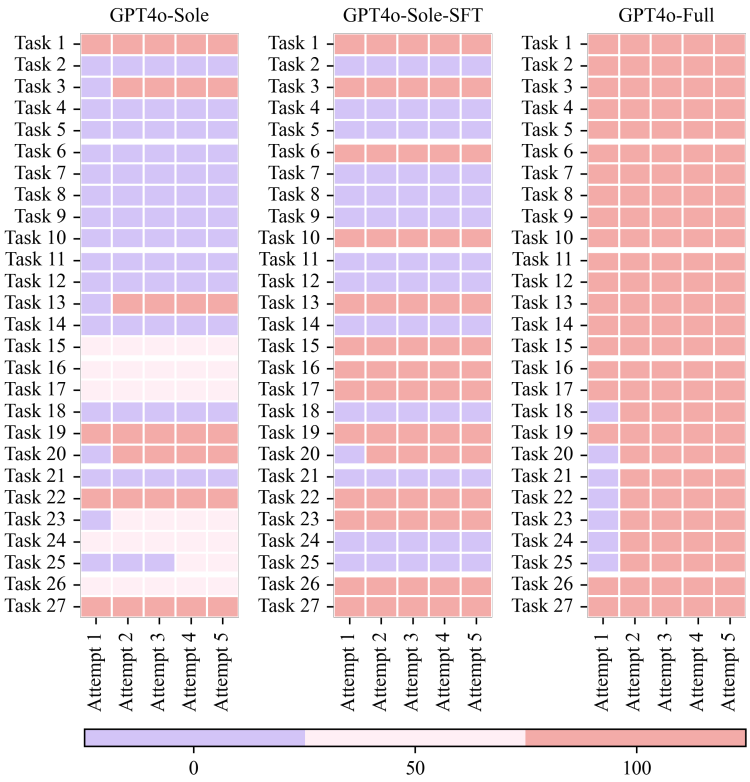

Fig. 12: Scores achieved by each evaluated scheme in individual attempts when handling standard tasks (Simulation Environment: MATPOWER; GPT4o version: gpt-4o-2024-08-06; open-source repository for all tasks and results: [link]).

based on the scores achieved in individual attempts on standard tasks. Evidently, SFT enhances the performance of `GPT4o-Sole`: `GPT4o-Sole-SFT` consistently attains more perfect scores (100 points) than `GPT4o-Sole`, with success rates of 51.11% versus 35.18%, respectively. However, `GPT4o-Full` significantly outperforms `GPT4o-Sole-SFT`, achieving 100 points for all tasks starting from the second attempt.

The underperformance of `GPT4o-Sole-SFT` could stem from the representativeness of the training datasets—despite significant human effort in generating them, these datasets may not and cannot capture every nuanced detail of MATPOWER. To rigorously eliminate the possibility that the limited dataset is responsible for the performance gap, an additional evaluation was conducted using the DALINE simulation environment. In this experiment, the full testing dataset (i.e., the DALINE simulation tasks used for evaluation) was used as the SFT training dataset. Specifically, all 34 testing simulation tasks, augmented with 16 additional tasks, were employed for fine-tuning, yielding a training dataset of 32,346 tokens (the dataset is available [here]). `GPT4o-Sole-SFT` was fine-tuned for 3 epochs, with a batch size of 1 and a learning rate multiplier of 2 (again, all automatically configured). The score distributions and specific success rates are presented in Fig. 13 and Table II. This evaluation shows that `GPT4o-Sole-SFT` improves the success rate of `GPT4o-Sole` from 0% to 51.96% overall, with 28.57% for complex tasks and 58.03%

for standard tasks. Yet, even with the entire testing dataset used for fine tuning, `GPT4o-Sole-SFT` still fails to reach high accuracy.

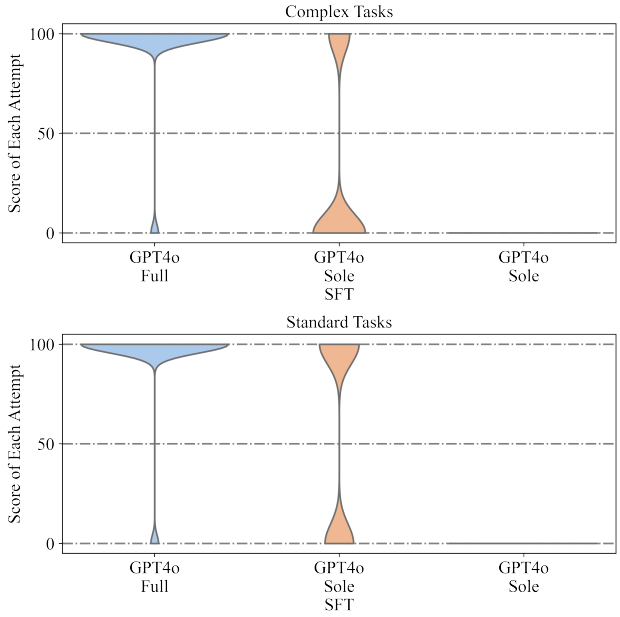

Fig. 13: Distribution of scores achieved across attempts for each evaluated scheme (Simulation Environment: DALINE; GPT4o version: gpt-4o-2024-08-06; open-source repository for all tasks and results: [link]).

TABLE II: Success rates for each scheme, broken down by all tasks combined, complex tasks only, standard tasks only, as well as for the first attempt success rate and the final attempt success rate (Simulation Environment: DALINE; GPT4o version: gpt-4o-2024-08-06; open-source repository for all tasks and results: [link])

Schemes	**GPT4o-Sole**	**GPT4o-Sole-SFT**	**GPT4o-Full**
All-Tasks	0.000%	51.961%	95.098%
All-Complex	0.000%	28.571%	95.238%
All-Standard	0.000%	58.025%	95.062%
First Attempt	0.000%	50.000%	91.176%
Final Attempt	0.000%	52.941%	97.059%

The underperformance of `GPT4o-Sole-SFT` can be attributed to the inherent limitations of SFT for LLMs: SFT can capture coarse-grained information such as style, tone, format, or other qualitative aspects, but SFT cannot memorize every nuance of the training data. The reason is twofold: (1) SFT is a lossy compression process, which inevitably discards finer details, and (2) SFT only adjusts a small subset of the model's parameters, with the rest remaining focused on generating generalizable patterns — as a result, SFT cannot achieve 100% retention of task-specific details, especially for tasks like coding, which require high-precision parameters and functions. In contrast, `GPT4o-Full` leverages an enhanced RAG module, allowing the model to dynamically access and retrieve precise information from external documents. This retrieval mechanism effectively mitigates the limitations of parameter-based memorization, enabling `GPT4o-Full` to achieve over 95% accuracy across both complex and standard tasks.

Abstract

...

It significantly outperforms ChatGPT 4o, o1-preview, and the fine-tuned GPT4o, which all achieved a success rate lower than 30% on complex tasks

...

Section VI. Conclusion

...

(iv) While SFT may be able to capture coarse-grained attributes such as style, tone, and format, it struggles to retain the full detail necessary for power system simulations. Even when the entire test dataset is included in the training set, SFT achieves a success rate of less than 60%. This limitation stems from its inherently lossy compression mechanism and constrained parameter update capacity, which prevent it from encoding fine-grained details with high fidelity.

...

Comment 2.2: Provide insights into the framework's adaptability to additional power system tools and tasks beyond the tested environments.

Response to Comment 2.2: First, in our preliminary preprint (unpublished work) $[5]_r$, only DALINE was used as an unseen simulation tool to evaluate the proposed framework understand various simulation tasks. This paper extends that work by incorporating MATPOWER into the evaluation, demonstrating the framework's adaptability across different simulation tools and tasks.

Second, to provide insights into the framework's adaptability, we have added a dedicated discussion on each module's adaptability at the end of its respective section. Detailed supplementary information is provided below.

Secion II. Enhanced RAG Module $\rightarrow$ II-C. Adaptability

The enhanced RAG module demonstrates high adaptability to a wide range of power system simulation tools. This flexibility is achieved by capturing the core principles of simulation coding—the use of functions and options. Specifically, the module decomposes simulation requests into distinct queries related to functions and options. This decomposition is tool-independent, thereby enabling the seamless integration of new simulation platforms. Additionally, the construction of a triple-based option document—linking options to their corresponding functions and dependencies—leverages inherent logical relationships that are common across different simulation tools. Together, these design choices underscore the module's robust adaptability in diverse simulation environments.

...

Secion III. Enhanced Reasoning Module $\rightarrow$ III-C. Adaptability

The enhanced reasoning module leverages few-shot chain-of-thought prompting techniques, which embody universal principles and remain tool-agnostic. Additionally, supported by the RAG module, the reasoning module integrates two levels of knowledge: the static and dynamic knowledge. The static

basic knowledge provides fundamental, coarse-grained insights—such as the basic principles of using a simulation tool. Meanwhile, dynamic retrieval knowledge supplies detailed, fine-grained information. This dual-layer approach is universally applicable across different tools. As a result, the enhanced reasoning module adapts seamlessly to a variety of simulation tasks and environments.

...

Secion IV. Environmental Acting Module with Feedback → IV-C. Adaptability

The environmental acting module features a tool-independent design. First, it leverages the error detection and reporting systems common to many power system simulation tools. Second, it formulates the error feedback loop in a general manner, as detailed in Section IV-B. Each component is independent of any specific simulation tool. This design not only supports automatic error correction but also improve adaptability to diverse simulation platforms.

Comment 2.3: Propose strategies for managing ambiguous or poorly structured error messages from simulation tools to improve robustness.

Response to Comment 2.3*:* In response, we have proposed strategies to manage ambiguous or poorly structured error messages, distinguishing between modifying internal error-reporting code and employing external strategies: First, the internal modification strategy utilizes our framework to systematically examine error reports across functional layers, identify ambiguities, and iteratively refine them, as noted in the original manuscript. This layered checking mechanism has been successfully implemented in DALINE, demonstrating high effectiveness. Second, for cases where internal modifications are impractical, we propose two external strategies: (i) batch extraction of relevant code segments for targeted analysis and (ii) an active trial-and-error mechanism that systematically introduces bugs to map ambiguous error messages. These strategies can be integrated into our framework to enhance robustness, with the trial-and-error mechanism offering a promising direction for future research.

For details, please refer to the added discussion in the paper:

Secion V. Case Study → V-B-4) First Attempt vs. Final Attempt

...

Enhancing error-reporting systems in simulation tools is beyond this paper's scope. However, for tools with ambiguous error messages, the following is proposed:

- **If internal modifications are feasible**, a layered checking mechanism can be implemented, where an agent evaluates error reports across functional layers, identifies vulnerabilities, and iteratively refines messages.
- **If internal modifications are not feasible**, three external strategies may be employed: (i) For LLMs with long-context capabilities, sending the entire underlying code for analysis is possible but costly. (ii) A more efficient alternative is to batch extract relevant code segments—from surface-layer to deeper-layer functions—and submit them alongside error messages to the agents for analysis. (iii) An active trial-and-error mechanism based on agents can automatically test code with intentional bugs,

mapping observed errors to ambiguous messages, in order to flag problematic outputs and facilitate learning.

These strategies can be seamlessly integrated into the proposed framework. Internal modifications require no changes to the framework since they occur within the simulation tool. Notably, the layered checking mechanism has already been implemented in DALINE, proving highly effective not only for the proposed framework but also as a robust enhancement for DALINE itself. For external strategies, the only adjustment needed is to extend the error report with relevant code extraction or error-message mapping. Future research will focus on leveraging the LLM-based trial-and-error mechanism to systematically map ambiguous error messages, reducing diagnosis overhead and improving robustness.

Comment 2.4: Enhance readability by shortening lengthy sentences and ensuring consistent terminology throughout the manuscript.

Response to Comment 2.4*:* First, we have restructured complex sentences for clarity and standardized expressions throughout the text. As the reviewer can see in the manuscript, many additional referential pronouns and transition words (highlighted in red) have been incorporated to reduce the length of sentences.

To ensure consistency in terminology, we have carefully reviewed key terms throughout the manuscript and made the following refinements:

- "Error feedback" $\rightarrow$ "Error-feedback" for uniformity in compound adjective usage.
- "Environmental Interaction Module" $\rightarrow$ "Environmental Acting Module" to maintain coherence with the terminology used in the framework description.
- "Power Systems Simulation" $\rightarrow$ "Power System Simulations" to align with the singular form conventionally used in academic writing.
- "Score" and "Success Rate" have been more clearly distinguished in the manuscript:
  - "Score" evaluates the correctness of each individual attempt for a given task under a specific scheme.
  - "Success Rate" aggregates these scores across multiple attempts and tasks to provide an overall measure of a scheme's performance.

Comment 2.5: Increase reliance on peer-reviewed sources to enhance the credibility of the research.

Response to Comment 2.5*:* First, we have updated the peer-reviewed status of the following studies in the revised manuscript, to enhance the credibility of our research:

- Reference $[6]_r$, published in the journal *Joule* in 2024.
- Reference $[7]_r$, published in the conference *NeurIPS* in 2024.
- Reference $[8]_r$, published in the conference *NeurIPS* in 2025.
- Reference $[9]_r$, published in the conference *IEEE PESGM* in 2024.
- Reference $[10]_r$, published in the conference *NeurIPS* in 2020.

Currently, aside from our preliminary findings in $[5]_r$, only three cited studies remain in their arXiv preprint status: [11]–$[13]_r$. These studies focus on the intersection of LLMs and power systems. Given that the integration

of LLMs into power systems is an emerging research direction, it is expected that some studies, such as [11]–[13]$_r$, are still under review. However, these references are cited solely to provide a broad yet concise overview of ongoing efforts in applying LLMs to power systems. More importantly, [11]–[13]$_r$ are unrelated to the methodology, case studies, or primary conclusions of this paper.

Comment 2.6: Minor: As an academic paper, please try to avoid using the first pronouns.

Response to Comment 2.6: Thank you for pointing this out. We have revised the manuscript to remove first-person pronouns throughout the manuscript and ensure a more formal academic tone.

REFERENCES

[1] M. Raghothaman, Y. Wei, and Y. Hamadi, "Swim: synthesizing what i mean: code search and idiomatic snippet synthesis," in Proceedings of the 38th International Conference on Software Engineering, 2016, pp. 357–367.

[2] X. Gu, H. Zhang, D. Zhang, and S. Kim, "Deep api learning," in Proceedings of the 2016 24th ACM SIGSOFT international symposium on foundations of software engineering, 2016, pp. 631–642.

[3] R. Agashe, S. Iyer, and L. Zettlemoyer, "Juice: A large scale distantly supervised dataset for open domain context-based code generation," arXiv preprint arXiv:1910.02216, 2019.

[4] E. Nijkamp, B. Pang, H. Hayashi, L. Tu, H. Wang, Y. Zhou, S. Savarese, and C. Xiong, "Codegen: An open large language model for code with multi-turn program synthesis," arXiv preprint arXiv:2203.13474, 2022.

[5] M. Jia, Z. Cui, and G. Hug, "Enabling large language models to perform power system simulations with previously unseen tools: A case of daline," arXiv preprint arXiv:2406.17215, 2024.

[6] S. Majumder, L. Dong, F. Doudi, Y. Cai, C. Tian, D. Kalathil, K. Ding, A. A. Thatte, N. Li, and L. Xie, "Exploring the capabilities and limitations of large language models in the electric energy sector," Joule, vol. 8, no. 6, pp. 1544–1549, 2024.

[7] H. Chang, J. Park, S. Ye, S. Yang, Y. Seo, D.-S. Chang, and M. Seo, "How do large language models acquire factual knowledge during pretraining?" in The Thirty-eighth Annual Conference on Neural Information Processing Systems, 2024.

[8] X. Wang, M. Feng, J. Qiu, J. Gu, and J. Zhao, "From news to forecast: Integrating event analysis in llm-based time series forecasting with reflection," Advances in Neural Information Processing Systems, vol. 37, pp. 58 118–58 153, 2025.

[9] C. Huang, S. Li, R. Liu, H. Wang, and Y. Chen, "Large foundation models for power systems," in 2024 IEEE Power & Energy Society General Meeting (PESGM). IEEE, 2024, pp. 1–5.

[10] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell et al., "Language models are few-shot learners," in Advances in neural information processing systems, vol. 33, 2020, pp. 1877–1901.

[11] B. Zhang, C. Li, G. Chen, and Z. Dong, "Large language model assisted optimal bidding of bess in fcas market: An ai-agent based approach," arXiv preprint arXiv:2406.00974, 2024.

[12] H. Wang, Z. Chen, N. Shang, S. Yao, Z. Pan, F. Wen, and J. Zhao, "Carbon footprint accounting driven by large language models and retrieval-augmented generation," arXiv preprint arXiv:2408.09713, 2024.

[13] X. Zhou, H. Zhao, Y. Cheng, Y. Cao, G. Liang, G. Liu, and J. Zhao, "Elecbench: a power dispatch evaluation benchmark for large language models," arXiv preprint arXiv:2407.05365, 2024.