MAEDA: An LLM-Powered Multi-Agent Evaluation Framework for EDA Tool Documentation QA

Abstract—**Large Language Models (LLMs) have shown remarkable capability in knowledge-intensive scenarios, such as electronic design automation (EDA) tool documentation question answering (QA), due to their ability to process and generate contextually rich, domain-specific information. Evaluating LLM outputs is paramount, as it directly impacts their accuracy, effectiveness, and trustworthiness in practical applications. In this paper, we introduce MAEDA, a novel LLM-powered multi-agent evaluation framework that utilizes multiple fine-tuned LLM agents working collaboratively to assess common error types encountered in EDA tool documentation QA. Specifically, we design customized point-to-point alignment and chain-of-thought (CoT) reasoning strategies tailored to specific agents, enhancing both fine-tuning and inference capabilities. Experimental results demonstrate that MAEDA outperforms state-of-the-art (SOTA) general-purpose and cross-domain evaluation frameworks in accurately identifying error types specific to this domain. Our benchmark is publicly available at https://anonymous.4open.science/r/MAEDA-DATE26.**

I. Introduction

With the continuous scaling of technology nodes and growing demands for performance and scalability, modern EDA tools have evolved into highly complex frameworks that support the entire design flow—from system-level specification to physical verification. Their intricate interdependencies result in voluminous documentation that is difficult to navigate, making it challenging and time-consuming for users to locate relevant content. Moreover, mastering the documentation requires substantial effort, with a steep learning curve and limited efficiency. Recent advances in LLMs, such as GPT-4 [1], Claude [2], and LLaMA3 [3], have led to their adaptation for various EDA-related [4], [5] tasks, including script generation [6], [7], HDL generation [8]–[17], verification [18]–[24], and QA [25]–[28]. In particular, EDA tool documentation QA aims to leverage LLMs to answer domain-specific questions grounded in tool documentation, thereby reducing the effort of manual documentation search and improving efficiency in using complex EDA tools. Building on this, Pu et al. [25], [29] introduce the RAGEDA framework, which integrates LLMs with retrieval-augmented generation (RAG) [30] to enhance answer reliability and accuracy by retrieving documentation content relevant to user queries and using it to augment the model's responses.

In EDA tool documentation QA, an effective evaluation framework is essential for identifying errors and guiding their iterative improvement. As illustrated in Fig. 1, such a framework should be capable of identifying hallucinations from the QA system's responses and providing feedback, which is then analyzed to refine specific modules of the system. Current EDA tool systems often rely on general-purpose metrics such as BLEU [31], ROUGE [32], and BERTScore [33], which primarily assess textual similarity but fail to capture specific error types critical in EDA scenarios, limiting their ability to provide meaningful insights for improving system performance.

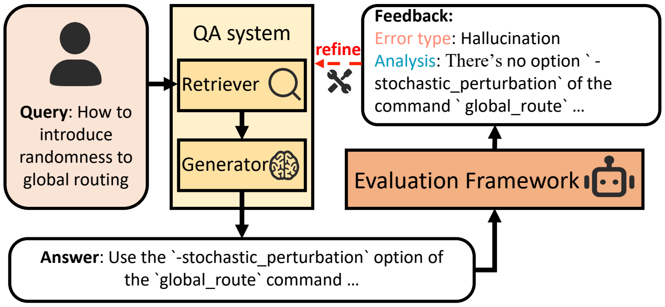

QA system
Retriever
Generator
refine
Query: How to introduce randomness to global routing
Feedback:
Error type: Hallucination
Analysis: There's no option `-stochastic_perturbation` of the command `global_route` ...
Evaluation Framework
Answer: Use the `-stochastic_perturbation` option of the `global_route` command ...

Fig. 1 An evaluation framework that detects QA errors and iteratively refines the system through error analysis.

Recent work on RAG evaluation has introduced innovative approaches such as RAGChecker's claim-level entailment evaluation [34], RAGEval's modular keypoint-based metrics [35], and RAGAS's reference-free paradigm [36]. Other methods like G-Eval [37] and DemoSG [38] standardize reasoning with structured CoT and schema-based constraints to enhance interpretability and reliability. However, these frameworks are designed for general-purpose use and rely mainly on score representations, limiting their ability to identify specific error types in specialized domains like EDA. This highlights the need for domain-specific evaluation frameworks for EDA tool documentation QA.

To address these challenges, we classify errors in EDA tool documentation QA into five categories: retrieval error, contradiction, missing, inappropriate refusal, and hallucination. We then introduce **MAEDA**, a multi-agent evaluation framework specifically designed for this task. In MAEDA, LLM-powered agents work collaboratively to identify diverse errors by employing structured CoT reasoning and a point-to-point alignment mechanism for precise error identification. In order to evaluate MAEDA and advance research in this domain, we introduce a benchmark of 300 QA pairs covering all defined error types. It contains both correct and erroneous QA cases, providing a comprehensive basis for assessing the accuracy and robustness of evaluation frameworks.

The main contributions of this work are as follows:

- We define five common error types in RAG-based EDA document QA, categorized into two main groups: Retrieval-level errors and Generator-level errors.

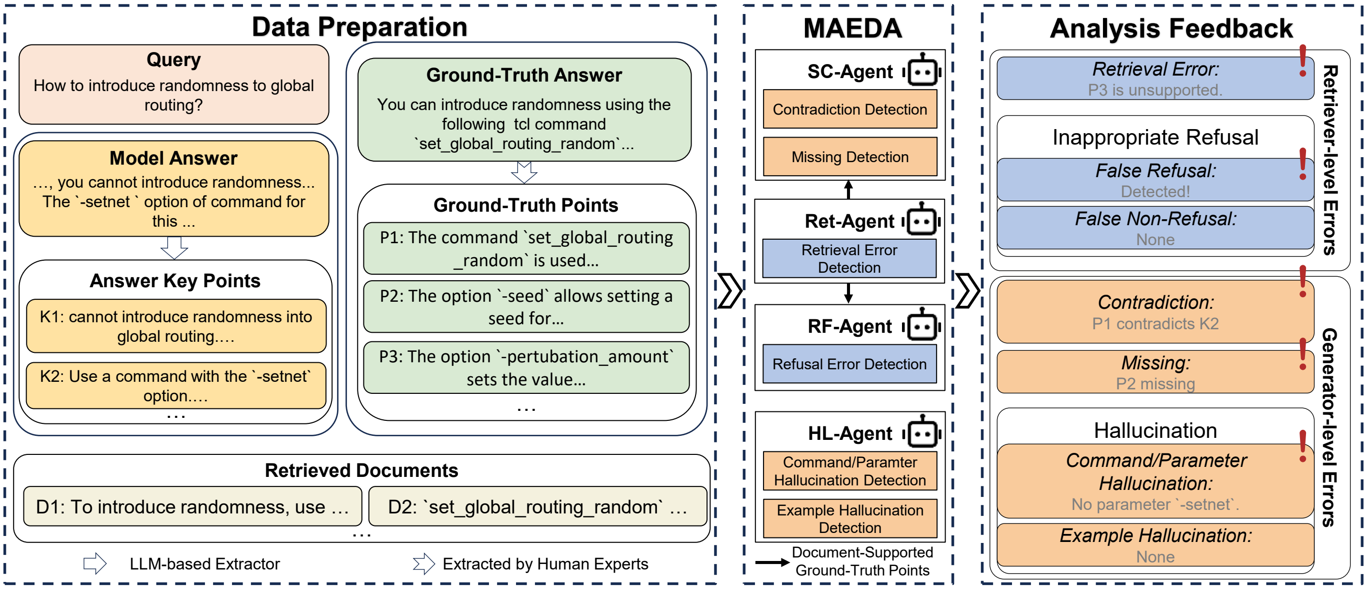

Data Preparation
Query
How to introduce randomness to global routing?
Model Answer
..., you cannot introduce randomness... The `-setnet ` option of command for this ...
Answer Key Points
K1: cannot introduce randomness into global routing....
K2: Use a command with the `-setnet` option....
...
Ground-Truth Answer
You can introduce randomness using the following tcl command `set_global_routing_random`...
Ground-Truth Points
P1: The command `set_global_routing _random` is used...
P2: The option `-seed` allows setting a seed for...
P3: The option `-pertubation_amount` sets the value...
...
Retrieved Documents
D1: To introduce randomness, use ...
D2: `set_global_routing_random` ...
...
LLM-based Extractor
Extracted by Human Experts
MAEDA
SC-Agent
Contradiction Detection
Missing Detection
Ret-Agent
Retrieval Error Detection
RF-Agent
Refusal Error Detection
HL-Agent
Command/Paramter Hallucination Detection
Example Hallucination Detection
Document-Supported Ground-Truth Points
Analysis Feedback
Retrieval Error:
P3 is unsupported.
Inappropriate Refusal
False Refusal:
Detected!
False Non-Refusal:
None
Retriever-level Errors
Contradiction:
P1 contradicts K2
Missing:
P2 missing
Hallucination
Command/Parameter Hallucination:
No parameter `-setnet`.
Example Hallucination:
None
Generator-level Errors

Fig. 2 Illustration of the proposed MAEDA framework.

- We propose MAEDA, a multi-agent evaluation framework for EDA tool documentation QA, leveraging structured CoT reasoning and a customized point-to-point alignment mechanism to assess diverse error types.
- We introduce a comprehensive evaluation benchmark, on which MAEDA outperforms SOTA general-purpose evaluation methods in identifying domain-specific errors.

II. Preliminaries

A. LLMs and RAG

LLMs are built on Transformer architectures with self-attention [39], which enable modeling of long-range dependencies and complex semantics, achieving strong performance in language understanding and generation.

RAG, widely applied in both open-domain and domain-specific QA, augments generation with retrieval to better address complex queries. A typical RAG framework consists of a retriever, a generator, and a domain-specific corpus. In recent years, RAG has shown strong performance across a range of tasks, including QA, code generation, and dialogue systems, and has been integrated into real-world systems like Bing Search and LangChain [40].

B. Problem Definition

The evaluation framework for EDA tool documentation QA requires four inputs: the user query (Q), the model-generated answer (A_{gen}), retrieved documents ($D = d_1, d_2, \ldots, d_n$), and the expert-annotated ground-truth answer (A_{gt}). Formally, the function of the evaluation framework is defined as:

$$\text{Eval}(Q, A_{\text{gen}}, D, A_{\text{gt}}) \rightarrow \{\text{err}_1, \text{err}_2, \ldots, \text{err}_n\}, \qquad (1)$$

where $\{\text{err}_1, \text{err}_2, \ldots, \text{err}_n\}$ represents the set of identified error types. These error types can be categorized into two levels: retriever-level and generator-level. (1) The retriever-level errors assess the relevance and sufficiency of retrieved documents for answer generation. (2) The generator-level errors assess the semantic quality of the generated answer. The objective is to enhance error type identification, with detailed definitions provided in Section III-A.

III. Methodology

A. Error Type Definition

Before detailing the error definitions, we first introduce symbolic notations. The ground-truth point set is denoted as $P = \{p_1, p_2, \ldots, p_l\}$. Each point is manually extracted from A_{gt} by human experts, representing a complete unit that contributes to answering the question, such as command usage, tool behavior, or summary conclusions. Similarly, the generated-answer key point set is denoted as $K = \{k_1, k_2, \ldots, k_m\}$, extracted from A_{gen} by an LLM-based extractor. A command (e.g., `place_design`) or parameter (e.g., `-max_cap`) in the documentation is denoted as c, and its usage example is denoted as e. From A_{gen}, we extract the command/parameter set $C = \{c_1, c_2, \ldots, c_x\}$ and example set $E = \{e_1, e_2, \ldots, e_y\}$ through rule-based extraction, while simultaneously extracting the command/parameter set $M = \{m_1, m_2, \ldots, m_k\}$ from the retrieved documents D. The following presents the definitions of each error type.

Retriever-level Errors. Determine whether the retrieved documents D contain any of the following errors.

Definition 1 (Retrieval Error)**.** *This error occurs when D lacks sufficient evidence to support A_{gt}, i.e., a ground-truth point $p_i \in P$ is not supported by any statement within D.*

Definition 2 (Inappropriate Refusal)**.** *Inappropriate Refusal can be subdivided into False Refusal and False Non-Refusal. False Refusal occurs when the model generates a refusal despite supporting evidence in D, i.e., A_{gen} refuses while A_{gt} provides an answer, and at least one statement in D*

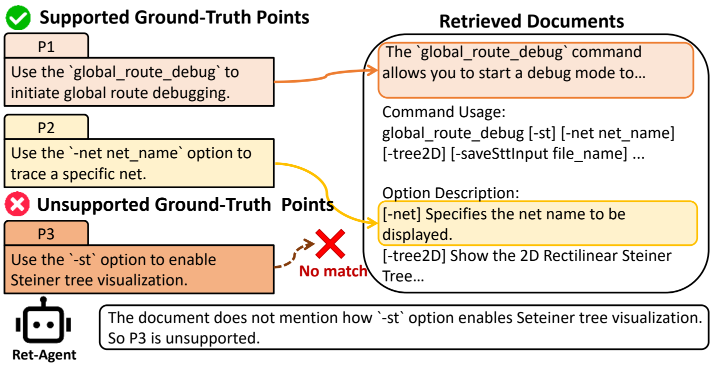

Supported Ground-Truth Points
P1
Use the `global_route_debug` to initiate global route debugging.
P2
Use the `-net net_name` option to trace a specific net.
Unsupported Ground-Truth Points
P3
Use the `-st` option to enable Steiner tree visualization.
No match
Retrieved Documents
The `global_route_debug` command allows you to start a debug mode to...
Command Usage:
global_route_debug [-st] [-net net_name] [-tree2D] [-saveSttInput file_name] ...
Option Description:
[-net] Specifies the net name to be displayed.
[-tree2D] Show the 2D Rectilinear Steiner Tree...
Ret-Agent
The document does not mention how `-st` option enables Seteiner tree visualization. So P3 is unsupported.

Fig. 3 An illustration of the Ret-Agent, which detects *retrieval errors* through structured reasoning and produces document-supported points for collaborative evaluation.

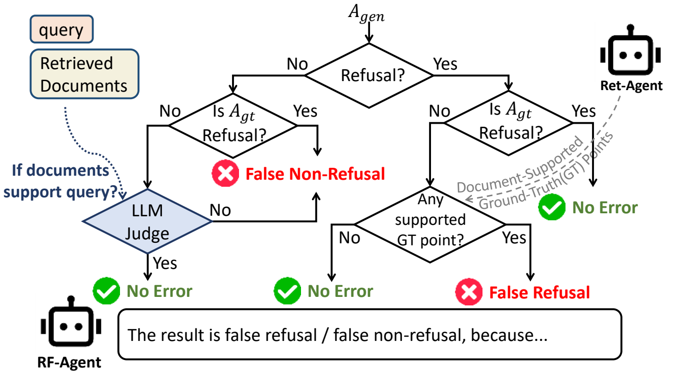


Fig. 4 Workflow of the RF-Agent, which compares the refusal behaviors of the generated and ground-truth answers, and collaborates with the Ret-Agent and LLM-based reasoning to detect *Inappropriate Refusal*.

supports the ground-truth point p_i. False Non-Refusal occurs when the model provides an answer to a question Q that is unanswerable from D, i.e., when both A_{gen} and A_{gt} provide answers, but D is semantically unrelated to Q.

Generator-level Errors. Determine whether the generated answer A_{gen} contains any of the following types of errors.

Definition 3 (Contradiction)**.** *This error occurs when A_{gen} conflicts with A_{gt}, i.e., a key point $k_j \in K$ semantically contradicts a ground-truth point $p_i \in P$.*

Definition 4 (Missing)**.** *This error occurs when ground-truth information is omitted in the A_{gen}, i.e., a ground-truth point $p_i \in P$ is not fully covered by any key point $k_j \in K$ in A_{gen}.*

Definition 5 (Hallucination)**.** *Hallucination can be subdivided into Command/Parameter Hallucination and Example Hallucination. Command/Parameter Hallucination occurs when A_{gen} fabricates commands or parameters that have no grounding in D, i.e., a generated command/ parameter $c_i \in C$ cannot be matched to either $m_j \in M$ extracted from D, or commands explicitly mentioned in Q. Example Hallucination occurs when a generated example $e_i \in E$ lacks grounding in D, i.e., an example $e_i \in E$ in A_{gen} cannot be matched to any usage explanation in D.*

B. Multi-Agent Evaluation Framework

Fig. 2 illustrates the overall flow of **MAEDA**, our customized evaluation framework for RAG-based EDA tool documentation QA system. Given a query, for example, “How to introduce randomness to global routing?”. The RAG system retrieves relevant documents D from the domain-specific corpus and generates an answer A_{gen}. Subsequently, both A_{gen} and A_{gt} are decomposed into fine-grained points. This point-wise representation enables precise alignment between model output and ground-truth answer.

MAEDA then employs a set of specialized agents to evaluate errors in the RAG system’s outputs. The Retrieval Error Identification Agent detects retrieval errors and derives document-supported ground-truth points, which are then leveraged by the Semantic Consistency Agent for contradiction and missing evaluation, ensuring assessments are unaffected by retriever deficiencies, and by the Refusal Evaluation Agent to assess the appropriateness of refusal behavior. Meanwhile, the Hallucination Evaluation Agent inspects A_{gen} for unsupported commands and examples. Working in coordination, these agents ensure comprehensive and consistent assessment, and their outputs are integrated in an analysis module that provides interpretable feedback for systematic error diagnosis and iterative model improvement. The following sections detail the structure and reasoning strategies of each agent module.

Retrieval Error Identification Agent (Ret-Agent). The first step in evaluating errors for EDA tool documentation QA is to assess the retriever by determining the correspondence between retrieved documents D and ground-truth answers A_{gt}. For this purpose, we define the retrieval error. However, we find that directly prompting LLMs to map this correspondence and identify retrieval errors are often ineffective. To address this, we develop the Ret-Agent with a domain-customized CoT reasoning strategy. The agent identifies ground-truth answer points fully supported by D and produces reliable structured output, which forms the basis for downstream error evaluation.

As illustrated in Fig. 3, the agent first identifies semantically relevant content in D for each ground-truth point to establish potential point-to-chunk matches, and then determine whether the matched content provides complete coverage and logical consistency. As shown in the figure, the agent identifies supporting document chunks for ground-truth points 1 and 2, which are passed to subsequent agents for semantic consistency and refusal evaluation. However, for ground-truth point 3, the document provides only an example of the parameter’s usage without any explanatory information, indicating a retrieval error in this case. To enhance robustness in structured reasoning and complex semantic alignment, we fine-tune the base model with supervision data derived from the agent’s structured reasoning process, enabling the model to better capture reasoning patterns in specific domain scenarios.

Refusal Evaluation Agent (RF-Agent). In EDA tool documentation QA, it is crucial for the QA system to refuse queries

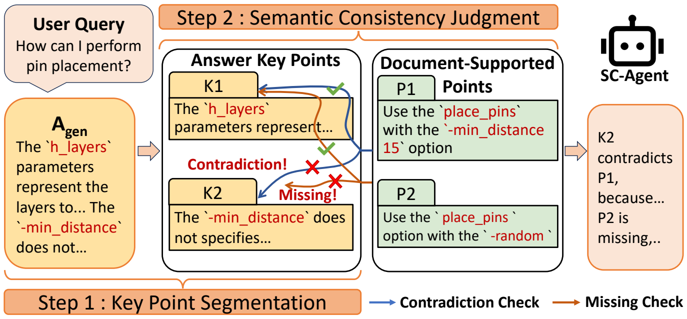

User Query
How can I perform pin placement?
Step 2 : Semantic Consistency Judgment
Answer Key Points
K1
The `h_layers` parameters represent...
Document-Supported Points
P1
Use the `place_pins` with the `-min_distance 15` option
SC-Agent
A_{gen}
The `h_layers` parameters represent the layers to... The `-min_distance` does not...
Contradiction!
Missing!
K2
The `-min_distance` does not specifies...
P2
Use the `place_pins` option with the `-random`
K2 contradicts P1, because... P2 is missing,...
Step 1 : Key Point Segmentation
Contradiction Check
Missing Check

Fig. 5 Overview of SC-Agent. It detects *Contradiction* and *Missing* through point-to-point alignment with document-supported ground-truth points.

beyond the scope of the retrieved documents D to ensure reliability. However, general LLMs often lack the structured reasoning capabilities required for domain-specific scenarios, limiting their ability to accurately evaluate refusal errors. To address this, we design a dedicated Refusal Evaluation Agent, which assesses whether the RAG system's refusal or non-refusal behavior is appropriate under D. This agent focuses on two types of retriever-level errors: (1) Incorrect Refusal and (2) Incorrect Non-Refusal.

The agent adopts a multi-branch reasoning structure (see Fig. 4). It first applies pattern-based matching to identify refusal behaviors in both A_{gen} and A_{gt}. In straightforward cases, such as when both answers are refusals (Correct Refusal) or only A_{gt} is a refusal (Incorrect Non-Refusal), the agent makes intuitive decisions guided by the pattern-based refusal classification. For more ambiguous situations, the agent employs LLM-based reasoning. When A_{gen} is a refusal but A_{gt} answers, the RF-Agent consults the Ret-Agent to determine whether any ground-truth point is document-supported. If such a point exists, the refusal is classified as Incorrect Refusal. In another case, when neither A_{gen} nor A_{gt} refuses, the agent checks whether retrieved documents are relevant to the query; if not, the case is classified as Incorrect Non-Refusal.

Semantic Consistency Agent (SC-Agent). Accurately identifying generator-level errors such as factual contradictions and missing information in A_{gen} is critical for evaluating the reliability of RAG systems and improving their generators in domain-specific QA tasks. However, both A_{gen} and A_{gt} often contain abundant and logically complex information, making direct prompting of LLMs unreliable for identifying these errors. To address this, we introduce the SC-Agent equipped with a domain-specific point-to-point alignment mechanism.

Guided by a structured prompt, the SC-Agent enables the LLM to perform multi-step alignment and classification reasoning. As shown in Fig. 5, the agent takes Q, A_{gen}, and document-supported points derived from the Ret-Agent as inputs. It first segments A_{gen} according to the format of the ground-truth points, extracting a set of key points for subsequent fine-grained alignment. Then, for each document-supported point, the agent performs two evaluations: (1) it checks whether any key point contradicts this ground-truth point (e.g., opposing conclusions or inconsistent premises), and (2) it detects whether any key point semantically covers the current ground-truth point. For instance, as illustrated in Fig. 5, the agent identifies a contradiction between key point 2 and ground-truth point 1, while ground-truth point 2 has no matching key point, indicating a missing error. These reasoning steps are output in a structured, multi-label format.

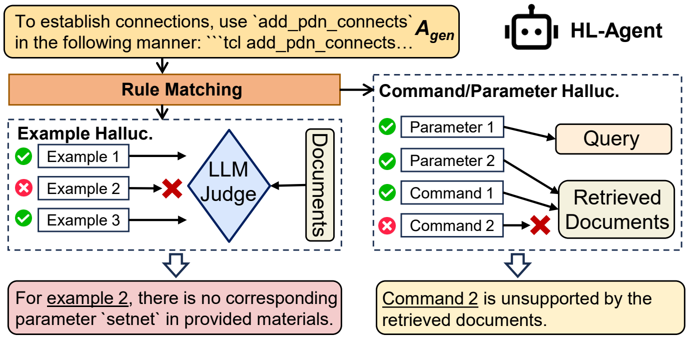

To establish connections, use `add_pdn_connects` in the following manner: ```tcl add_pdn_connects...
A_{gen}
HL-Agent
Rule Matching
Command/Parameter Halluc.
Example Halluc.
Example 1
Example 2
Example 3
LLM Judge
Documents
Parameter 1
Parameter 2
Command 1
Command 2
Query
Retrieved Documents
For example 2, there is no corresponding parameter `setnet` in provided materials.
Command 2 is unsupported by the retrieved documents.

Fig. 6 An illustration of the HL-Agent for *hallucination* detection, where Command/Parameter Hallucination is identified by rule-based matching and Example Hallucination by rule-based extraction with LLM-based reasoning.

By leveraging the point-to-point alignment, the Semantic Consistency Agent achieves fine-grained identification of contradictions and missing errors, thereby grounding the evaluation in explicit semantic correspondence rather than coarse-grained heuristics. To improve the accuracy and stability of structured reasoning in complex semantic tasks, we fine-tune the base model with instruction tuning.

Hallucination Evaluation Agent (HL-Agent). In the highly specialized task of EDA tool documentation QA, A_{gen} may contain hallucinations, such as fabricated commands, parameters, or invalid usage examples. General-purpose LLMs are largely ineffective at identifying these errors due to limited domain knowledge, making a domain-specific hallucination detection mechanism essential. To this end, we design an agent that addresses two common types of hallucination: Command/-Parameter Hallucination and Example Hallucination. To reliably identify these errors, we employ a rule-based extraction mechanism tailored to the formatting specifications of EDA tool commands and their usage examples. The evaluation flow is shown in Fig. 6, with details provided below:

For Command/Parameter Hallucination, the agent extracts well-formatted commands and parameters from A_{gen} and compares them against two valid sources: the user query Q and the retrieved documents D. Any command or parameter extracted from A_{gen} that does not match these sources is classified as a hallucination. As shown in Fig. 6, command 2 has no matching command in either Q or D, so it is identified as a hallucination. The rule-based design ensures efficiency and reliability in detecting such errors.

For Example Hallucination, we design a hybrid process combining the extraction mechanism with LLM-based reasoning. Regular expressions and format constraints first extract standardized examples from A_{gen}. Then, an LLM guided by task-specific structured prompts compares them with the usage guidelines in D, ensuring compliance with command and parameter specifications. Any undefined parameter or violation of parameter value constraint is identified as an example hallucination. As shown in Example 2 of Fig. 6, an undefined parameter is used, resulting in an example hallucination.

C. Model Fine-Tuning

This subsection introduces the instruction tuning approach [41]. Given parameters θ and an input sequence $X = \{x_1, x_2, \ldots, x_n\}$, the probability of generating X is formulated as:

$$P(X; \theta) = \prod_{t=1}^{n} P(x_t \mid x_{<t}; \theta). \tag{2}$$

The training objective is to minimize the negative log-likelihood loss:

$$\mathcal{L}(\theta) = -\sum_{t=1}^{n} \log P(x_t \mid x_{<t}; \theta), \tag{3}$$

where θ denotes the trainable parameters.

IV. Benchmark

Existing open-source QA evaluation benchmarks (e.g., RAGAS, RAGBench [42]) primarily focus on general-purpose tasks, failing to capture domain-specific error types in EDA. Therefore, there is an urgent need to develop a high-quality benchmark tailored to evaluating the RAG system in EDA documentation QA. To validate our multi-agent evaluation framework, we construct a dataset for fine-grained error type identification, which is open-source at https://anonymous.4open.science/r/MAEDA-DATE26.

Our benchmark is constructed based on the open-source EDA QA dataset, ORD-QA [25], which is developed from OpenROAD [43], an open-source system that provides a complete RTL-to-GDSII digital design flow. The dataset includes four categories: functionality, VLSI-flow, GUI, and installation & test, covering the standard functional modules and usage scenarios of EDA tools. To construct this benchmark, two approaches are adopted: First, we leverage real cases collected from a RAG-based workflow, which naturally captures practical retrieval and generation behaviors in EDA tasks. Second, inspired by OmniEval [44], we design an automated data construction pipeline tailored to EDA QA. GPT-4o is guided by carefully crafted prompts to generate reliable negative samples by modifying specific content according to particular error types. Additionally, ground-truth points are manually identified by domain experts from A_{gt}, enabling fine-grained evaluation. All generated samples are manually reviewed to ensure quality and reliability. The final benchmark comprises 300 QA instances, including both positive and negative cases, covering all defined error types.

V. Experiments

A. Experimental Setup

In our MAEDA framework, instruction tuning is conducted on the Qwen3-14B [45], using QLoRA [46] for parameter-efficient fine-tuning. The model is trained for 2 epochs with a learning rate of 1e-4 and a batch size of 16. Optimization uses the paged_adamw_32bit optimizer with cosine learning rate scheduling. LoRA is configured with rank of 32, alpha of 16, and dropout rate of 0.05. The model is fine-tuned on an A100 GPU with 40GB memory.

For baselines, three representative frameworks are selected: **RAGChecker** [34], **RAGEval** [35] and **G-Eval** [37]. For RAGChecker, the Claim Recall with a threshold of 0.7 is used to detect retrieval errors, any non-zero Hallucination score is considered as hallucination. For RAGEval, Retriever Recall with the same threshold of 0.7 is applied to identify retrieval errors. A Completeness score below 0.8 indicates a missing error, any Hallucination score greater than zero is considered a hallucination, and an Irrelevance score exceeding 0.1 indicates contradiction. For G-Eval, GPT-4o is employed to automatically generate the evaluation CoT for each metric, and three outputs are sampled to estimate token probabilities.

B. Training Dataset Collection

This subsection discribes the dataset construction for model fine-tuning in MAEDA. We fine-tune two models, one for Ret-Agent and the other for SC-Agent. Inspired by the strategy in RAGEDA, the OpenROAD documentation is segmented into paragraph chunks, from which coherent chunks R are extracted. Based on R, GPT-4o generates a query Q, and performs step-by-step reasoning to produce A_{gt}, which is then paraphrased to obtain the corresponding A_{gen}. Finally, fine-grained answer points P are extracted from A_{gt}.

Following this process, we construct 3,197 positive samples as five-tuples $\{Q,\ R,\ A_{\text{gt}},\ A_{\text{gen}},\ P\}$. Negative samples are generated by modifying content with tailored prompts for different error types. Both positive and negative samples are then evaluated within MAEDA using DeepSeek-R1 [47], which generates the supervision signals S. After manual filtering, the final dataset is represented as six-tuples $\{Q,\ R,\ A_{\text{gt}},\ A_{\text{gen}},\ P,\ S\}$, comprising 2,094 instances for retrieval error evaluation and 2,670 for contradiction and missing evaluation.

C. Evaluation Results

To evaluate the effectiveness of our framework, we compare Qwen3-14B, GPT-4o, and our fine-tuned models with baselines on the 300 QA instances constructed in Section IV. To eliminate interference, each agent is evaluated independently. The results are shown in TABLE I, presenting recall and precision for each error type.

In the **Retrieval Error Identification** task, MAEDA with our fine-tuned model performs comparably to GPT-4o and surpasses all baselines, demonstrating strong capability in document identification and semantic matching for knowledge-intensive EDA tasks. In the more challenging tasks of **Contradiction** and **Missing**, our fine-tuned model outperforms GPT-4o with recall/precision of 0.941 / 0.800 and 0.928 / 0.726, indicating that structured reasoning supervision signals are effective in enhancing complex semantic evaluation. For the **Inappropriate Refusal**, MAEDA achieves a recall of 0.987 across all settings, far surpassing G-Eval and demonstrating robustness across different models. In the domain-specific

TABLE I Model performance on fine-grained error type classification (on OpenROAD Dataset)

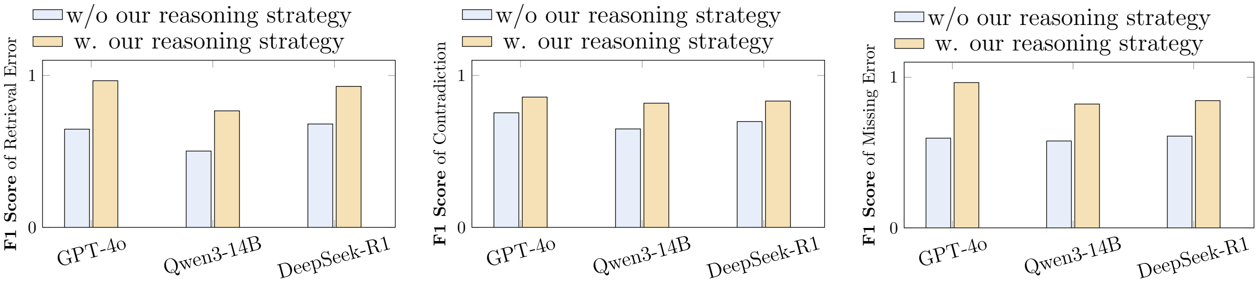
Method	Retrieval Error		Contradiction		Missing		Inappropriate Refusal		Hallucination	
	Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision
RAGChecker [34]	0.960	0.692	–		–		–		0.486	0.346
RAGEval [35]	0.958	0.926	0.059	0.039	0.825	0.650	–		0.095	0.132
G-Eval [37]	0.703	0.703	0.881	0.394	0.772	0.550	0.267	0.143	0.385	0.294
MAEDA w. Qwen3-14B	0.747	0.789	0.902	0.676	0.887	0.601	**0.987**	0.711	**0.935**	0.680
MAEDA w. GPT-4o	0.960	**0.947**	0.922	0.758	0.907	0.633	**0.987**	**0.757**	0.860	**0.769**
MAEDA w. Fine-tuned models*	**0.973**	0.936	**0.941**	**0.800**	**0.928**	**0.726**	**0.987**	0.711	**0.935**	0.680

* denotes the use of our fine-tuned models for retrieval error, contradiction and missing, with Qwen3-14B for other error types.

TABLE II Model performance on fine-grained error type classification (on Commercial EDA Tool Dataset)

Method	Retrieval Error		Contradiction		Missing		Inappropriate Refusal		Hallucination	
	Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision
RAGChecker [34]	0.821	0.902	–		–		–		0.648	0.246
RAGEval [35]	0.857	0.528	0.033	0.056	0.788	0.788	–		0.038	0.125
G-Eval [37]	0.833	0.641	0.476	0.217	0.852	0.264	0.667	0.08	0.546	0.270
MAEDA w. Qwen3-14B	0.597	**0.976**	0.667	0.625	0.818	0.600	0.827	**0.811**	**0.872**	0.531
MAEDA w. GPT-4o	0.866	0.951	0.767	0.697	0.848	**0.800**	**0.940**	0.758	0.782	**0.550**
MAEDA w. Commercial tool models†	**0.896**	0.952	**0.800**	**0.727**	0.879	0.725	0.827	**0.811**	**0.872**	0.531
MAEDA w. Fine-tuned models*	0.866	0.921	0.700	0.700	**0.909**	0.600	0.827	**0.811**	**0.872**	0.531

† denotes the use of commercial EDA tool models, similarly trained, for retrieval error, contradiction and missing, with Qwen3-14B for other error types.



(a) Retrieval Error

(b) Contradiction

(c) Missing

Fig. 7 Ablation study on the impact of our reasoning strategy across three error types (i.e., Retrieval Error, Contradiction, and Missing) and different LLMs (i.e., GPT-4o, Qwen3-14B, and DeepSeek-R1).

task of **Hallucination Evaluation**, MAEDA with Qwen3-14B performs comparably to GPT-4o and far surpasses the three baselines, demonstrating the utility of rule-based matching and the robustness of the reasoning strategy.

Furthermore, to verify the transferability of our framework, we construct a documentation QA dataset for a commercial EDA tool (timing ECO platform), which follows the same standard as the OpenROAD dataset and comprises 205 QA instances. Results in TABLE II show the outperformance of our framework compared with the baselines, confirming that MAEDA generalizes effectively to different EDA scenarios.

D. Ablation Study

In this subsection, we conduct ablation experiments to assess the impact of structured CoT reasoning and fine-grained semantic alignment within MAEDA. Three representative LLMs (GPT-4o, Qwen3-14B, and DeepSeek-R1) are compared with and without our reasoning strategy on Retrieval Error, Contradiction, and Missing. F1 score is adopted as the evaluation metric, reflecting a balanced measure of performance.

The results in Fig. 7 demonstrate consistent F1 improvements with our strategy across models and error types. For example, the F1 score of GPT-4o on Retrieval error increases significantly from 0.6462 to 0.9535, indicating that even advanced commercial models have reasoning limitations in domain-specific tasks. These improvements confirm that explicit reasoning pathways and fine-grained alignment substantially enhance semantic evaluation, confirming both the effectiveness and generalizability of the proposed approach.

VI. Conclusion

We propose MAEDA, a multi-agent evaluation framework for EDA tool documentation QA. By defining domain-specific error types and integrating structured CoT reasoning with point-to-point semantic alignment, MAEDA provides accurate and interpretable error identification. To support further research, we release an OpenROAD-based benchmark for documentation QA evaluation. In addition, we develop tailored training strategies for model fine-tuning to enhance domain-specific evaluation with open-source models. Experiments on our datasets show that MAEDA outperforms SOTA methods.

References

[1] J. Achiam, S. Adler, S. Agarwal *et al.*, "GPT-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[2] Y. Bai, A. Jones, K. Ndousse *et al.*, "Training a helpful and harmless assistant with reinforcement learning from human feedback," *arXiv preprint arXiv:2204.05862*, 2022.

[3] A. Grattafiori, A. Dubey, A. Jauhri *et al.*, "The llama 3 herd of models," *arXiv preprint arXiv:2407.21783*, 2024.

[4] Z. He, Y. Pu, H. Wu, Y. Qiu, T. Qiu, B. Yu *et al.*, "Large language models for eda: From assistants to agents," *Foundations and Trends® in Electronic Design Automation*, vol. 14, no. 4, pp. 295–314, 2025.

[5] Z. He, Y. Pu, H. Wu, T. Qiu, and B. Yu, "Large language models for eda: Future or mirage?" *ACM Transactions on Design Automation of Electronic Systems*.

[6] H. Wu, Z. He, X. Zhang *et al.*, "ChatEDA: A large language model powered autonomous agent for eda," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[7] Z. He, H. Wu, X. Zhang *et al.*, "ChatEDA: A Large Language Model Powered Autonomous Agent for EDA," in *ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*, 2023, pp. 1–6.

[8] M. Liu, T.-D. Ene, R. Kirby *et al.*, "Chipnemo: Domain-adapted LLMs for chip design," *arXiv preprint arXiv:2311.00176*, 2023.

[9] K. Chang, Y. Wang *et al.*, "ChipGPT: How far are we from natural language hardware design," *arXiv preprint arXiv:2305.14019*, 2023.

[10] M. Liu, N. Pinckney *et al.*, "VerilogEval: Evaluating large language models for verilog code generation," in *IEEE/ACM International Conference on Computer Aided Design*, 2023, pp. 1–8.

[11] Y. Fu, Y. Zhang, Z. Yu *et al.*, "Gpt4aigchip: Towards next-generation ai accelerator design automation via large language models," in *IEEE/ACM International Conference on Computer Aided Design*, 2023, pp. 1–9.

[12] J. Blocklove, S. Garg, R. Karri, and H. Pearce, "Chip-Chat: Challenges and opportunities in conversational hardware design," in *ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*, 2023, pp. 1–6.

[13] S. Thakur, J. Blocklove, H. Pearce, B. Tan, S. Garg, and R. Karri, "AutoChip: Automating HDL generation using LLM feedback," *arXiv preprint arXiv:2311.04887*, 2023.

[14] Y. Lu, S. Liu, Q. Zhang *et al.*, "RTLLM: An open-source benchmark for design RTL generation with large language model," in *Asia and South Pacific Design Automation Conference*, 2024, pp. 722–727.

[15] S. Thakur, B. Ahmad, Z. Fan *et al.*, "Benchmarking large language models for automated Verilog RTL code generation," in *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2023, pp. 1–6.

[16] S. Thakur, B. Ahmad, H. Pearce *et al.*, "Verigen: A large language model for Verilog code generation," *ACM Transactions on Design Automation of Electronic Systems*, vol. 29, no. 3, pp. 1–31, 2024.

[17] Z. Pei, H.-L. Zhen *et al.*, "Betterv: Controlled verilog generation with discriminative guidance," *arXiv preprint arXiv:2402.03375*, 2024.

[18] Y. Tsai, M. Liu, and H. Ren, "RTLFixer: Automatically fixing RTL syntax errors with large language model," in *ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[19] M. Orenes-Vera, A. Manocha, D. Wentzlaff *et al.*, "Autosva: Democratizing formal verification of RTL module interactions," in *ACM/IEEE Design Automation Conference (DAC)*, 2021, pp. 535–540.

[20] R. Kande, H. Pearce, B. Tan *et al.*, "LLM-assisted generation of hardware assertions," *arXiv e-prints*, pp. arXiv–2306, 2023.

[21] X. Meng *et al.*, "Unlocking hardware security assurance: The potential of LLMs," *arXiv preprint arXiv:2308.11042*, 2023.

[22] S. Paria, A. Dasgupta, and S. Bhunia, "Divas: An llm-based end-to-end framework for soc security analysis and policy-based protection," *arXiv preprint arXiv:2308.06932*, 2023.

[23] B. Ahmad, S. Thakur, B. Tan, R. Karri, and H. Pearce, "Fixing hardware security bugs with large language models," *arXiv preprint arXiv:2302.01215*, 2023.

[24] X. Yao, H. Li, T. H. Chan *et al.*, "HDLdebugger: Streamlining HDL debugging with large language models," *ACM Transactions on Design Automation of Electronic Systems*, 2024.

[25] Y. Pu, Z. He, T. Qiu *et al.*, "Customized retrieval augmented generation and benchmarking for EDA tool documentation QA," in *IEEE/ACM International Conference on Computer-Aided Design*, 2024, pp. 1–9.

[26] U. Sharma, B.-Y. Wu *et al.*, "OpenROAD-Assistant: An Open-Source Large Language Model for Physical Design Tasks," in *ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*, 2024, pp. 1–7.

[27] A. Kaintura, S. S. Luar, I. I. Almeida *et al.*, "ORAssistant: A Custom RAG-based Conversational Assistant for OpenROAD," 2024.

[28] B.-Y. Wu, U. Sharma *et al.*, "EDA Corpus: A Large Language Model Dataset for Enhanced Interaction with OpenROAD," 2024.

[29] Y. Pu *et al.*, "Customized Retrieval Augmented Generation and Benchmarking for EDA Tool Documentation QA," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2025.

[30] P. Lewis, E. Perez, A. Piktus *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," *Advances in neural information processing systems*, vol. 33, pp. 9459–9474, 2020.

[31] K. Papineni *et al.*, "Bleu: a method for automatic evaluation of machine translation," in *Annual Meeting of the Association for Computational Linguistics*, 2002, pp. 311–318.

[32] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Text summarization branches out*, 2004, pp. 74–81.

[33] T. Zhang, V. Kishore *et al.*, "Bertscore: Evaluating text generation with bert," *arXiv preprint arXiv:1904.09675*, 2019.

[34] D. Ru *et al.*, "Ragchecker: A fine-grained framework for diagnosing retrieval-augmented generation," *Advances in Neural Information Processing Systems*, vol. 37, pp. 21 999–22 027, 2024.

[35] K. Zhu, Y. Luo, D. Xu, Y. Yan, Z. Liu, S. Yu, R. Wang, S. Wang, Y. Li, N. Zhang *et al.*, "Rageval: Scenario specific rag evaluation dataset generation framework," *arXiv preprint arXiv:2408.01262*, 2024.

[36] S. Es *et al.*, "Ragas: Automated evaluation of retrieval augmented generation," in *Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, 2024, pp. 150–158.

[37] Y. Liu, D. Iter, Y. Xu *et al.*, "G-eval: Nlg evaluation using gpt-4 with better human alignment," *arXiv preprint arXiv:2303.16634*, 2023.

[38] G. Zhao, X. Gong, X. Yang *et al.*, "Demosg: Demonstration-enhanced schema-guided generation for low-resource event extraction," *arXiv preprint arXiv:2310.10481*, 2023.

[39] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[40] H. Chase, "Langchain, october 2022," *URL https://github.com/langchain-ai/langchain*, 2022.

[41] J. Wei, M. Bosma, V. Y. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le, "Finetuned language models are zero-shot learners," 2022. [Online]. Available: https://arxiv.org/abs/2109.01652

[42] R. Friel, M. Belyi, and A. Sanyal, "Ragbench: Explainable benchmark for retrieval-augmented generation systems," *arXiv preprint arXiv:2407.11005*, 2024.

[43] T. Ajayi, V. A. Chhabria, M. Fogaça *et al.*, "Toward an open-source digital flow: First learnings from the openroad project," in *Proceedings of Annual Design Automation Conference 2019*, 2019, pp. 1–4.

[44] S. Wang, J. Tan, Z. Dou, and J.-R. Wen, "Omnieval: An omnidirectional and automatic rag evaluation benchmark in financial domain," 2025. [Online]. Available: https://arxiv.org/abs/2412.13018

[45] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv *et al.*, "Qwen3 technical report," *arXiv preprint arXiv:2505.09388*, 2025.

[46] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," *Advances in neural information processing systems*, vol. 36, pp. 10 088–10 115, 2023.

[47] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi *et al.*, "Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning," *arXiv preprint arXiv:2501.12948*, 2025.