Customized Retrieval Augmented Generation and Benchmarking for EDA Tool Documentation QA

Yuan Pu, Zhuolun He, Yuqi Jiang, Tairu Qiu, Haoyuan Wu, Qi Sun, *Member, IEEE*, Cheng Zhuo, *Senior Member, IEEE*, and Bei Yu, *Senior Member, IEEE*

Abstract*—Retrieval augmented generation (RAG) improves the accuracy and dependability of generative AI models by integrating factual information from external databases. This technique is widely used in tasks involving document-grounded question answering (QA). While these RAG systems are extensively pretrained on general-purpose documents, they face considerable limitations when applied to specialized, knowledge-intensive fields such as electronic design automation (EDA). This article addresses such issue by proposing a customized RAG framework along with three domain-specific techniques for EDA tool documentation QA, including a contrastive learning scheme for text embedding model fine-tuning, a reranker distilled from proprietary large language model (LLM), and a generative LLM fine-tuned with high-quality domain corpus. To further unleash the extraordinary language capacity of LLMs in the domain of EDA-tool documentation QA, we propose to train LLMs as the reranker model with our customized two-stage traning scheme, which consists of the point-wise instruction tuning stage and the pair-wise learn-to-rank (LTR) stage. Finally, we have developed and released a documentation QA evaluation benchmark, ORD-QA, for OpenROAD, an advanced RTL-to-GDSII design platform. Experimental results demonstrate that our proposed RAG flow and techniques have achieved superior performance on ORD-QA as well as on a commercial tool, compared with state-of-the-arts. Furthermore, compared with the SOTA reranker models, our LLM reranker prominently improves the document retrieval accuracy and thus leads to better QA quality. The ORD-QA benchmark and the training dataset for our customized RAG flow are open-source at https://github.com/lesliepy99/RAG-EDA.

***Index Terms*—Design automation, information retrieval, large language models, retrieval augmented generation.**

I. Introduction

Electronic design automation (EDA) encompasses a suite of software tools specifically aimed at designing, analyzing, and verifying electronic systems. These tools are essential to address the growing demands of semiconductor manufacturing and to support emerging technology nodes. They facilitate a complex design process by offering a myriad of functionalities at various stages. Open-source EDA tools, such as OpenROAD [1] and iEDA [2], deliver a comprehensive RTL-to-GDSII flow, enabling robust environment for circuit design. Meanwhile, commercial software incorporates a broader spectrum of advanced functionalities, meticulously tailored for industrial applications. EDA tools are supposed to be equipped with well-organized and detailed documentations, yet referring documentation often remains a pain for end users due to the huge amount of functionalities provided and the intertanglement among them. To enhance the user experience and development efficiency for tool users, vendors employ and train application engineers to offer on-site customer support. However, the substantial costs associated with training and manpower have led to considerations of implementing automated, no-human-in-the-loop systems.

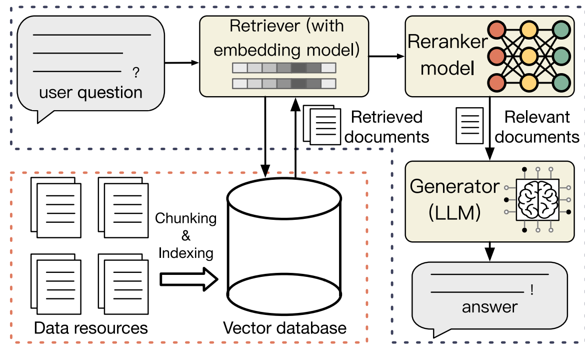

Fig. 1. Illustration of the RAG flow.

The remarkable performance and rapid advancement of large language models (LLMs) have demonstrated that retrieval augmented generation (RAG) is a potent method for document-grounded question answering (QA). As is shown in Fig. 1, to prepare for RAG, the entire document is segmented into reasonably sized chunks, each of which is then embedded into a high-dimensional space by a text embedding model and stored in a vector database. When a user poses a question, the text embedding model encodes the question into an embedding in the same space. Through similarity matching, document chunks that most closely match the semantics of the question are retrieved as potential reference candidates. Then, a reranker model conducts fine filtering and removes irrelevant document chunks from the candidates. Finally, the relevant document

Received 12 October 2024; revised 22 January 2025 and 12 April 2025; accepted 3 May 2025. Date of publication 9 May 2025; date of current version 24 November 2025. This work was supported in part by The Research Grants Council of Hong Kong, SAR, under Grant CUHK14210723; in part by The National Natural Science Foundation of China under Grant 62304197; and in part by the MIND Project under Grant MINDXZ202404. This article was recommended by Associate Editor J. R. Doppa. *(Corresponding author: Bei Yu.)*

Yuan Pu, Haoyuan Wu, and Bei Yu are with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, SAR (e-mail: byu@cse.cuhk.edu.hk).

Zhuolun He and Tairu Qiu are with the Research and Development Department, ChatEDA Tech, Foshan 528200, China.

Yuqi Jiang, Qi Sun, and Cheng Zhuo are with the College of Integrated Circuits, Zhejiang University, Hangzhou 310058, China.

Digital Object Identifier 10.1109/TCAD.2025.3568776

chunks along with the user query are fed to the generator (usually an LLM) for answer generation.

The RAG process effectively utilizes the reasoning and inference capabilities of LLMs while retrieving relevant information to reduce hallucinations and improve the reliability of generated outputs. However, applying existing RAG flows to EDA tool documentation QA faces significant challenges due to the complex structure of EDA documentation, the intricate nature of EDA flows, and the rigorous logical reasoning required for EDA-related queries. The primary challenge stems from a lack of expert EDA knowledge in current RAG components, resulting in suboptimal performance in EDA-specific contexts: First, current text embedding models used in the retriever component exhibit inadequate semantic understanding of EDA terminologies and concepts, often leading to the retrieval of irrelevant documents. Second, existing reranker models fail to accurately differentiate between relevant and weakly related documents under EDA-tool-specific scenarios. Lastly, the generators, typically chat LLMs, lack domain-specific knowledge in EDA and show limited logical inference abilities when tackling EDA-related questions, resulting in ineffective answer generation. With the rapid development of LLM, many previous research work have customized LLMs for several EDA tasks, including script generation [3], [4], HDL generation [5], [6], [7], [8], [9], [10], [11], [12], [13], [14], code verification and analysis [15], [16], [17], [18], [19], [20], [21], and EDA tool documentation QA [22], [23], [24]. Among these, while OpenROAD-Assistant [23] and ORAssistant [24] employ the RAG framework, they rely on general-purpose retriever or reranker models rather than fine-tuning domain-specific models. This approach results in lower document retrieval accuracy when addressing complex user queries related to EDA tools.

To address the aforementioned challenges, we introduce *RAG-EDA*, a tailored RAG flow specifically designed for EDA tool documentation QA. This approach includes three key enhancements.

1) For retriever, we begin by developing a corpus that integrates EDA domain knowledge and terminologies. This corpus is utilized to finetune a text embedding model through contrastive learning, thereby enriching it with EDA-specific knowledge and enhancing its retrieval accuracy.
2) For reranker, we distill the fine filtering capabilities of the proprietary LLM into our reranker model by contrastive learning, greatly improving its performance in weakly related documents removal.
3) For generator, we pretrain a chat LLM using the EDA domain corpus and subsequently finetune it using a dataset of EDA-tool-related QA pairs through instruction tuning, enhancing its capability to generate accurate and contextually relevant answers.

Furthermore, existing reranker models, either lacking EDA domain knowledge or relying on small language models, struggle to capture the deep logical relationships between queries and documents, leading to poor performance in filtering out weakly related documents. To address this, we leverage the advanced language capabilities of LLMs as the reranker model and introduce a customized two-stage training scheme: 1) pointwise instruction tuning in the first stage and 2) pairwise learn-to-rank (LTR) in the second stage.

Finally, to assess the efficacy of our RAG flow and facilitate future research in EDA tool documentation QA, we have developed and made public a QA evaluation dataset, *ORD-QA*, based on the OpenROAD documentation. This dataset comprises 90 high-quality question-document-answer triplets that span a variety of query types, including VLSI flow inquiries, command/option usage, installation guides, and GUI usage. To further assess the robustness, universality, and transferability of our proposed RAG-EDA flow, we have developed an internal QA evaluation dataset based on the documentation of a commercial EDA tool.

Our major contributions are summarized as follows.

1) We carefully analyze the limitations of applying off-the-shelf RAG to EDA tool documentation QA, and propose a customized RAG flow to address them accordingly. Our solution outperforms for both academic and commercial EDA tools.
2) We conduct contrastive learning and incorporate EDA expertise to finetune the text embedding model and the reranker model, achieving superior retrieval performance than SOTA retrievers.
3) For generator training, we propose a two-stage training scheme, consisting of the domain-knowledge pretrain and task-specific instruction tuning.
4) We propose to train a LLM as the reranker model, and customize a two-stage training scheme, including the pointwise instruction tuning stage and the pairwise learn to rank stage.
5) We release a QA benchmark with 90 high-quality QA pairs concerning OpenROAD, to evaluate our RAG flow and facilitate the future research on EDA tool documentation QA.

II. Preliminaries

A. Information Retrieval

Given the user query, information retrieval (IR) is the process of obtaining information from a collection of documents or data sources that are relevant to the query. In the RAG, IR usually consists of two stages: 1) retrieval (coarse-grained) and 2) reranking (fine-grained).

Retrieval (Coarse-Grained IR): Retrieval methods can be categorized into two types: 1) lexical (sparse) retrieval and 2) semantic (dense) retrieval. In lexical retrieval, the query is tokenized into keywords, and techniques like TF-IDF or BM25 are used to rank document chunks based on keyword relevance. The top-k chunks with the highest scores are selected as relevant documents. Semantic retrieval uses a pretrained text embedding model to map the query and document chunks into a shared semantic embedding space. The top-k chunks closest to the query in this vector space, representing the most semantically similar content, are selected [25].

While semantic retrieval generally outperforms lexical methods by capturing semantic meanings, it independently

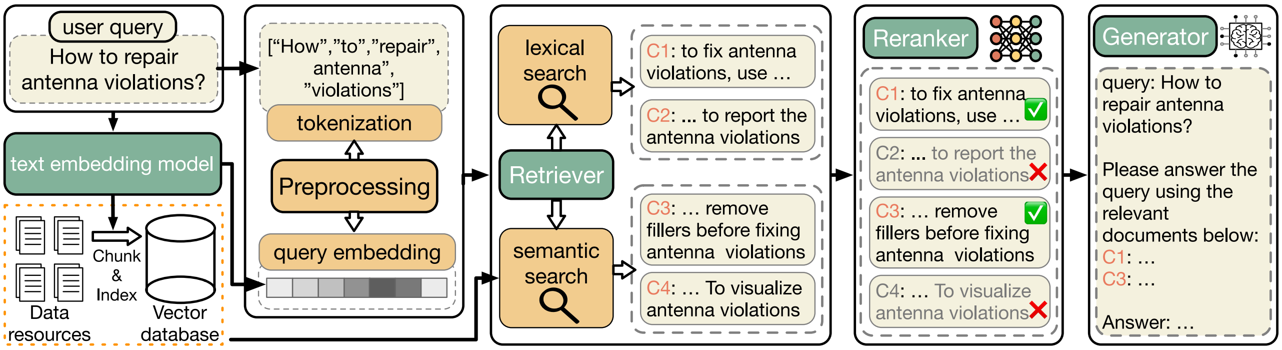

Fig. 2. Overview of RAG-EDA, our proposed RAG flow for EDA tool documentation QA.

encodes queries and documents, ignoring mutual logical relationships. This can reduce its effectiveness in filtering weakly related documents. To address this, fine-grained techniques like RRF [26] and reranking are often applied after coarse-grained retrieval to enhance accuracy.

Reranking (Fine-Grained IR): At the fine-grained reranking stage, reranker models typically use the cross-encoder architecture [27]. During inference, the model processes the concatenated query-document pair as input and outputs a scalar relevance score. By encoding the pair in a single pass, the model captures deep interactions and mutual logical relationships, leading to improved IR accuracy.

B. Performance Measurement

To assess a retriever/reranker's effectiveness in retrieving relevant documents and filtering out irrelevant ones, recall is used as the metric. Recall@k is mathematically defined as the proportion of relevant documents retrieved among the top k results. Let R represent the set of all relevant documents in the corpus, and R_k denote the relevant documents within the top k results. Then, recall@k is formulated as

$$\text{recall@}k = \frac{|R_k|}{|R|}. \tag{1}$$

To evaluate the performance of a generator in answering EDA-tool-related questions, we use three widely recognized metrics: 1) BLEU [28]; 2) ROUGE-L [29]; and 3) UniEval [30]. BLEU, originally designed for machine translation, is now widely applied in text generation evaluation. ROUGE-L measures the similarity between the LLM-generated answer and the reference answer by identifying the longest common subsequence (LCS). UniEval is a comprehensive LLM-based evaluation framework that assesses generated text across dimensions, such as factual consistency, relevance, and coherence. For EDA tool QA, we specifically use UniEval's factual consistency score, which ranges from 0 to 1, with higher scores indicating better accuracy.

III. RAG-EDA

Fig. 2 presents the overall process of *RAG-EDA*, our customized retrieval-augmented generation (RAG) approach for EDA tool documentation QA. In the preparation phase, the EDA documentation is divided into manageable chunks and encoded into vectors using a customized text embedding model (Section III-A) to build a vector database. The first stage, preprocessing, tokenizes the user query and encodes it into an embedding. The second stage, retrieval (Section III-B), combines lexical retrieval (e.g., TF-IDF or BM25) with semantic retrieval from the vector database to generate document candidates. Next, the finetuned reranker model (Section III-C) filters the candidates, removing weakly related documents. Finally, the refined document chunks, along with the query, are passed to the generator model (Section III-D) to produce the answer. The following sections detail the customization and techniques used at each stage.

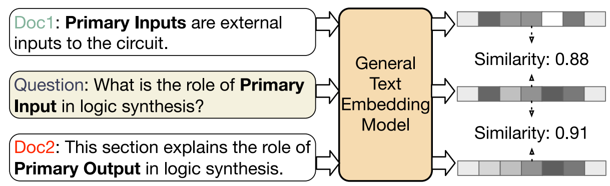

Fig. 3. Fail case of general text embedding model in EDA-specific IR (about PI).

A. Domain-Customized Text Embedding

In semantic retrieval, the text embedding model plays a critical role by converting user queries and document chunks into numerical vectors that capture their semantic meaning. The quality of these embeddings directly impacts retrieval effectiveness. While many proprietary and open-source embedding models [27], [31] perform well in general tasks, they often lack sufficient understanding of EDA-specific terminology and concepts. As shown in Fig. 3, even state-of-the-art models struggle to distinguish between basic EDA terms like Primary Input (PI) and Primary Output (PO). This leads to inaccuracies in the semantic vector space, causing the selection of less relevant documents. For example, Doc2 might be chosen over the more relevant Doc1 due to incorrect semantic proximity, reducing the QA flow's overall effectiveness.

To resolve the identified issue and boost the retriever's performance, we implement supervised contrastive learning to refine the model's accurate recognition of EDA terminologies and concepts. The fundamental principle of our contrastive

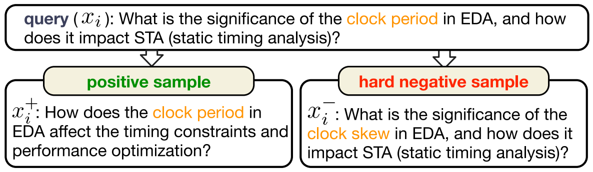

Fig. 4. Contrastive data sample used for text embedding model finetuning.

learning approach is as follows: If two sentences involve the same EDA terminology, they should be positioned closely in the embedded vector space, despite differences in sentence structures. Conversely, if two sentences have similar structures but pertain to different EDA terminologies or concepts, they should be distanced apart in the embedding space. Specifically, for one EDA-domain user query x_i, we denote its positive and hard negative samples by x_i^+ and x_i^-, respectively. The positive sample x_i^+ involves the same EDA term with the query x_i, meanwhile the negative sample x_i^- has a similar sentence structure with the query, but pertains to different EDA terminology/concept. Fig. 4 shows one example of the triplet $\{x_i, x_i^+, x_i^-\}$: the query x_i and the positive sample x_i^+ are user questions about "clock period," while the hard negative sample x_i^- share the same sentence structure with x_i but contains a different EDA terminology (clock skew). We construct a triplet dataset $\{x_i, x_i^+, x_i^-\}_{i=1}^N$ of size N. Then, we apply mini-batch gradient descent and supervised contrastive learning [32], [33] to finetune the embedding model. Assume the mini-batch size is M, and for each query x_i in the mini-batch, we apply in-batch sampling. The contrastive loss of s_i can be written as

$$-\log \frac{e^{\mathrm{sim}\left(x_i, x_i^+\right)/\tau}}{\sum_{j=1}^{M}\left(e^{\mathrm{sim}\left(x_i, x_j^+\right)/\tau} + e^{\mathrm{sim}\left(x_i, x_j^-\right)/\tau}\right)} \quad (2)$$

where *sim* stands for the cosine similarity function, τ is the temperature hyperparameter.

B. Hybrid Information Retrieval

In EDA tool documentation QA, user queries span categories, such as VLSI flow inquiries, command/option usage, installation guidelines, GUI usage, and test unit, each with distinct syntactic and structural features. Lexical retrieval methods like TF-IDF or BM25 perform well when queries include specific keywords or phrases directly matching the documentation. In contrast, semantic retrieval is more effective for queries lacking explicit keywords, such as those related to VLSI flow. Building on the insights gained from the analysis above, our approach diverges from the traditional method of exclusively using semantic search for IR [34]. In our proposed framework, we adopt a hybrid search strategy that combines both semantic and lexical search techniques to retrieve documents. The document chunks retrieved through this hybrid approach are then merged and deduplicated to create a pool of relevant document candidates. Subsequently, candidate documents that are weakly related to the query will be further differentiated and excluded at the reranking stage, as detailed in Section III-C.

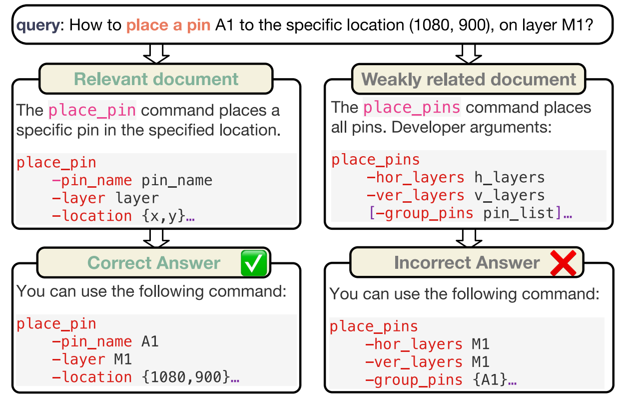

Fig. 5. Weakly related documents do harm to answer generation quality for EDA-tool questions.

C. Reranker Finetuning

Document chunks retrieved by the hybrid retriever either share semantic similarities with the query or contain matching keywords/phrases. These chunks are classified into two categories.

1) Relevant documents that directly address the query, and
2) Weakly related documents that share similar EDA terminology or concepts but cannot provide a direct answer.

Research [35] shows that weakly related documents can lead augmented LLMs to generate inaccurate responses. As illustrated in Fig. 5, a query about the command to place a specific pin retrieves the relevant document for *place_pin*, resulting in a correct answer. However, referencing a weakly related document about placing all pins leads to an incorrect response.

Reranking is a widely adopted technique for filtering out weakly related or irrelevant documents. State-of-the-art rerankers in RAG typically employ a cross-encoder architecture, which jointly encodes the query and each candidate document using cross-attention to compute relevance scores. The top-k documents with the highest scores are then selected as the most relevant. While cross-encoder models like bge-reranker perform well on general IR tasks, they are less effective at distinguishing between relevant and weakly related documents in the context of EDA tool documentation QA. In contrast, we observe that proprietary LLMs, such as GPT-4, when guided by carefully designed prompts, can effectively filter out weakly related documents in EDA-related QA. However, their autoregressive nature leads to slower inference, and the high deployment cost remains a limiting factor.

To bridge the gap between existing cross-encoder reranker models and proprietary LLMs in the context of EDA tool QA documents filtering, we initiate our approach by collecting a set of EDA-tool-related questions, denoted as $Q = \{q_1, q_2, \ldots, q_n\}$. For each $q_i \in Q$, we apply the hybrid retriever described in Section III-B to retrieve k candidate documents C_i. We then employ GPT-4 with task-specific prompt to differentiate relevant documents (C_i^+) and weakly related documents (C_i^-), ensuring $|C_i^+| + |C_i^-| = k$. Next, to improve the reranker model's capability to differentiate between relevant documents and weakly related ones, even

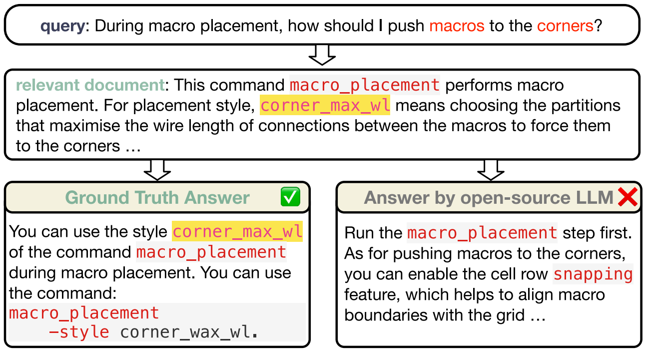

Fig. 6. Fail case of existing open-source chat LLMs for answering the EDA-tool questions.

when they share similar semantics or identical keywords with the query, we implement a contrastive learning scheme. This scheme fine-tunes the reranker model using documents that have been prefiltered by GPT-4. Specifically, for a question $q_i \in Q$, we sample one positive document s_i^+ from C_i^+ and m negative documents $\{s_{i,1}^-, s_{i,2}^-, \ldots, s_{i,m}^-\}$ from C_i^- $(m \leq k)$, the contrastive loss for q_i can be then written as

$$-\log \frac{e^{f(q_i,s_i^+)/\tau}}{e^{f(q_i,s_i^+)/\tau} + \sum_{j=1}^{m} e^{f\left(q_i,s_{i,j}^-\right)/\tau}} \tag{3}$$

where τ denotes the temperature, $f(q, s)$ denotes the scalar score of the query-document pair (q, s) outputted by the reranker model, and the negative sample size m is set to 3.

D. Domain-Specific LLM Generator

In the final stage of the RAG flow, the generator, typically a LLM, processes the user query and the filtered relevant document(s) to produce an answer. Our observations indicate that directly employing open-source chat LLMs as generators for EDA tool documentation QA results in a lower quality of answers. Fig. 6 shows one typical example: The user inquires for the instruction to push macros to the corner during macro placement, the ground-truth answer to this question is to enable the *corner_max_wl* style of the command *macro_placement*, which is depicted in the relevant document. When using an open-source chat LLM, namely, Qwen-14B-chat, the generated answer is erroneous. Although some proprietary LLMs, such as GPT-4 [36] demonstrate extraordinary capability in problem solving, due to the privacy of the end user information, a locally deployed LLM is preferred out of the consideration of security. Therefore, it is necessary to finetune and deploy a local LLM as the generator in the EDA tool QA system.

By analysis, we get the conclusion that there are mainly two limitations of existing chat LLMs for EDA tool documentation QA.

1) Both Open-source and proprietary LLMs lack specialized knowledge in EDA. Questions concerning EDA tool usage are often complex, and while the provided relevant documents are useful, additional domain-specific knowledge can be crucial for answering queries effectively. For example, besides the description of the command *macro_placement*, the domain knowledge of VLSI macro placement is also helpful to answer the question in Fig. 6.
2) General chat LLMs are ineffective in handling the questions involved with EDA tools. Answering such questions requires intricate and domain-specific analysis between the question and provided documents, and step-by-step inference, which are not well mastered by existing chat LLMs.

While many prior studies have explored enhancing LLMs with domain knowledge, such as self-correction [37], [38] and reinforcement learning (RL) techniques [39], [40], [41], these methods fail to address both limitations in our scenario simultaneously. On the one hand, these techniques mainly focus on improving the domain-specific inference capability of the models, and are not applicable for incorporating domain-related background knowledge. On the other hand, for both the self-correct methods and the RL techniques, it is very difficult to customize an accurate value function/reward function under the scenario of EDA tool documentation QA, due to the sophisticated nature of the EDA-tool-related questions and the nonuniqueness of the corresponding answers. Recently, there are several research works on EDA-specific LLM customization [5], [13], which splits the training process into the pretraining and fine-tuning stages to overcome the two limitations mentioned above. However, either their training corpus is not open-source or their domain customization is about RTL generation. To train a domain-customized LLM generator for EDA tool QA, we collect and open-source our customized training corpus, and propose a two-stage training scheme. The first stage is domain knowledge pretrain: To enhance the domain-specific knowledge of the chat LLM, we utilize two authoritative textbooks in EDA, specifically *EDA for IC Implementation, Circuit Design, and Process Technology* [42] and *Handbook of Algorithms for Physical Automation* [43], as the pretraining corpus. We segment these textbooks into paragraph-sized chunks to construct a dataset. Subsequently, we pretrain the chat LLM on this dataset using an autoregressive approach. The second stage is task-specific instruction tuning: To begin with, we leverage GPT4 to generate a set of triplets $\{q, r, a\}$ of high quality. For each triplet, q is one question related to the usage of the EDA tool, r is/are the document(s) used to answer the question, and a is the ground-truth answer. Then, we integrate a well-designed system prompt to each triplet to form a QA dataset. Finally, we finetune the pretrained chat LLM in stage 1 on the collected QA dataset.

It is important to recognize that LLMs operate in an autoregressive manner, where the prediction of each token is sequentially determined and depends on the tokens that have been previously generated. Given an LLM with parameter θ and a token sequence $x = \{x_1, x_2, \ldots, x_n\}$, the next-token prediction objective of training is to minimize the negative log-likelihood loss formulated in

$$L = -\sum_{t=1}^{n} \log P(x_t|x_{<t}; \theta). \tag{4}$$

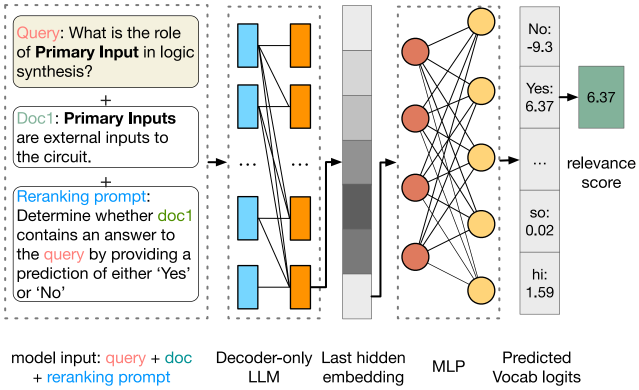

Fig. 7. Model architecture of the LLM reranker.

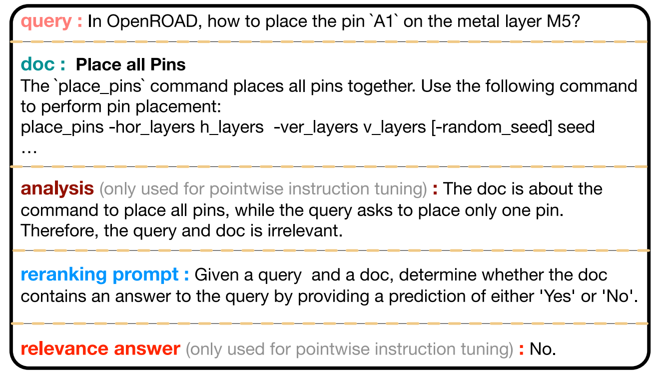

Fig. 8. Prompt template used for both the pointwise instruction tuning and the pairwise learn-to-rank training stages.

The two stages, namely, domain-knowledge pretrain and task-specific instruction tuning, are trained using the loss function defined in (4), on their respective corpus dataset.

IV. LLM Reranker

In this section, we first introduce the model architecture adopted by the LLM reranker. Then, the customized two-stage training scheme is introduced: At the first pointwise instruction tuning stage, the LLM reranker model is trained to analyze the relevance logical relationship between one query and one document, following which the second pairwise LTR stage trains the model to distinguish the relevance degrees among a set of documents given one query.

A. Architecture of LLM Reranker

We adopts the model architecture of the BGE reranker [27] for our propsoed LLM reranker, which conducts the cross encoder mechanism and consists of two modules: 1) the decoder-only LLM and 2) the multilayer perceptron (MLP) layer. Fig. 7 visualizes the model architecture of the LLM reranker. The decoder-only LLM adopts the casual attention mask, taking the concatenation of the user query, the document chunk and the tailored reranking prompt as input. The last hidden embedding of the last token, which captures the semantic and logical relavance relationship between the query and the document, is then fed to the MLP layer to predict the logits of the token vocabulary. The logits value of the "Yes" token is finally adopted as the relevance score, reflecting the relevance degree between the user query and the document.

B. Pointwise Instruction Tuning Stage

By observation, directly applying previous methodologies of reranking on EDA tool documentation QA faces one major challenge: existing reranker models or approaches lack domain knowledge in EDA, and therefore demonstrate poor performance in evaluating the relevance relationship between user queries and tool documents in EDA domain. Taking one typical example of the query "How to place one pin on layer M5?" in OpenROAD, the query requires to place one specific pin on the designated metal layer, and its relevant and weakly related documents are the "place_pin" command and the "place_pins" command, respectively. However, lacking the domain knowledge about the pin placement flow in the physical design flow, existing reranker models will literally analyze the query and documents, and assign similarly high scores to these two documents, worsening the ultimate answer generation quality.

To equip the LLM reranker model with EDA domain knowledge and augment the model's capacity of analyzing the relevance relationship between EDA-related questions and documents, we propose the pointwise instruction tuning technique, as our first training stage. Fig. 8 instantiates the prompt template used for the pointwise instruction tuning stage: Following the user query and the document, the "analysis" part includes the logical inference process for the relevance relationship between the query and the document. Then, the "reranking prompt" guides the model to predict whether the query and the document is relevant or not. Finally, the "relevance answer" outputs "Yes" or "No" for the final relevance determination. The term "pointwise" emphasizes that during instruction tuning, the logical analysis process between the user query and exactly one document is used as the training corpus. By providing the pointwise "analysis" part in the training corpus, the capacity of the LLM model in analyzing the logical relevance relationship between the EDA-domain query and document is well trained and improved. The fine-tuning adopts the autoregressive loss function defined in (4).

C. Pairwise Learn-to-Rank Training Stage

After the pointwise instruction tuning stage, the LLM reranker gains the ability to analyze the relevance relationship between a query-document pair. However, it still needs training to rank the relevance levels of multiple documents for a single query. Traditional rerankers are typically trained using a contrastive learning strategy with cross-entropy loss (3), where relevant documents are treated as positive samples and weakly related/irrelevant ones as negative samples. In EDA tool documentation QA, the relevance among a query's relevant documents can vary—some documents are more relevant and should be ranked higher. Contrastive learning, however, treats all relevant/weakly related documents equally as positive/negative samples, ignoring these relevance differences. This limitation results in poor reranking accuracy for complex queries and documents.

To account for differences in document relevance during training, we adopt the pairwise Learning-to-Rank (LTR) technique [44]. This method compares the relevance of each adjacent pair of documents in a list ranked by descending relevance for loss calculation. The pairwise LTR formulation is derived as follows: For a query q and two candidate documents x_i and x_j, let $\overline{P_{ij}}$ represent the ground-truth relevance ranking: $\overline{P_{ij}} = 1$ if x_i is more relevant than x_j, $\overline{P_{ij}} = 0$ if x_j is more relevant, and $\overline{P_{ij}} = 0.5$ if both have equal relevance. Denote the predicted relevance score for q and x_i by s_i, the predicted probability that x_i is more relevant to q than x_j can be written as

$$P_{ij} = \frac{e^{(s_i - s_j)}}{1 + e^{(s_i - s_j)}}. \tag{5}$$

By (5), the value difference of relevance scores of x_i and x_j is mapped to the value range between 0 and 1. The closer P_{ij} is to 1, the more relevant x_i is to the query q compared with x_j. Finally, the objective of pairwise LTR is to minimize the cross entropy between the ground truth and predicted probability, as is defined in

$$L = -\sum_{(i,j)} \overline{P_{ij}} \log P_{ij} + \left(1 - \overline{P_{ij}}\right) \log\left(1 - P_{ij}\right). \tag{6}$$

Given one query and a list of documents ranking in the descending order of relevance degrees, pairwise LTR compares the relevance degrees between each adjacent pair of documents utilizing (5), and uses (6) for loss function calculation. However, directly applying the adjacency pair sampling strategy in the traditional pairwise LTR technique to EDA tool document reranking faces one major challenge: The traditional pairwise LTR technique emphasizes on optimizing the ranking result of the whole document list, meanwhile for our scenario, we only care about the proportion of relevant documents among the top k results returned by the reranker model. Directly applying the adjacency pair sampling strategy wastes training resources and performance in ranking documents outside the top k reranking window (the weakly related and irrelevant documents), harming the performance of relevant document retrieval.

To overcome the above issue and emphasize the training process on relevant document retrieval, we propose two customized pair sampling strategies, namely, high-relevance adjacency pairing and one-to-all pairing. Before delving into the details of these two pair sampling strategies, the structure of the training data will first be introduced. Each item of the training data contains a given user query q and its corresponding candidate document lists D_q, each document $d_i \in D_q$ is assigned with a relevance score which ranges from 0 to 100 and the higher the relevance score is, the more relevance d_i is to q. We first sort D_q in the descending order of relevance scores to obtain the sorted document list D'_q. We name the first half of D'_q as the high-relevance document list, as documents in the first-half of D'_q are more relevant to q compared with the second half.

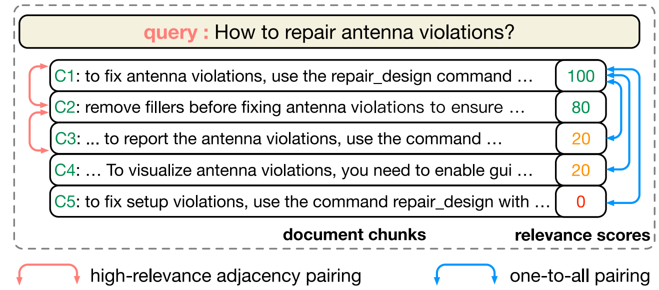

Fig. 9. Illustration of the pair sampling strategies. The list of document chunks is sorted in the descending order of relevance scores.

For high-relevance adjacency pairing strategy, the adjacent document pairs in the high-relevance document list are sampled for pairwise LTR, shifting the focus of the training process on high-relevance document retrieval. For the one-to-all pairing strategy, one document pair is sampled between the document with the highest relevance score and any other document chunk. By such sampling strategy, the training process focuses on distinguishing between the most relevant document and other weakly related/irrelevance documents, improving its performance on high-relevance document fine-filtering. The orange/blue double-sided arrows in Fig. 9 illustrates the high-relevance adjacency/one-to-all pairing strategy.

V. Benchmark

The EDA-Corpus [22] provides a high-quality Question-Answer benchmark consisting of 196 unique QA pairs. In practical scenarios involving EDA tool documentation QA, user queries often span multiple aspects of the documentation and require complex logical reasoning to resolve. However, most of the questions in EDA-Corpus are confined to a single document chunk, with their answers being directly and literally extractable from the corresponding content. This characteristic limits its ability to effectively evaluate the logical inference capabilities of the LLM generator in a customized RAG flow. There is a clear need for an EDA tool documentation QA benchmark where questions require referencing multiple document chunks and are logically sophisticated. Such a benchmark would necessitate logical inference that integrates both EDA domain knowledge and document content to generate accurate answers. To address this need, we developed and released the *ORD-QA* dataset. ORD-QA comprises 90 high-quality question-document-answer triplets derived from the OpenROAD [1] documentation. To create the dataset, the OpenROAD documentation[1] was divided into 290 segments based on section and section markdowns. One to five document chunks were then randomly selected at a time. For single chunks or logically related chunks, GPT-4 was used to generate EDA-related questions relevant to the selected content. Corresponding answers were generated based on the document chunks. Each triplet underwent thorough manual review to correct logical inconsistencies and errors. Additionally, realistic user queries about OpenROAD were

[1] https://openroad.readthedocs.io/en/latest/

manually crafted, with accurate answers provided to enrich the dataset.

The questions within the ORD-QA dataset are classified into four distinct categories: 1) functionality; 2) VLSI-flow; 3) GUI; and 4) installation & test. Functionality questions elaborate on the application of specific commands or options within particular user scenarios. VLSI-flow questions seek detailed steps required to execute a specific VLSI process. GUI questions pertain to the operation of the OpenROAD GUI, whereas installation & test questions encompass guidelines for software installation and queries related to unit test modules. For the distribution of question categories in ORD-QA, the questions of functionality, vlsi-flow, GUI, and install&test account for 51.1%, 24.4%, 18.9%, and 5.6%, respectively. 53.3% and 28.9% of the questions are formulated based on one and two document chunks. The rest (17.8%) are based on more than three document chunks. Among the 90 questions, 89 of them refer to no more than five document chunks, with only one question based on seven document chunks. Compared to the EDA-Corpus QA benchmark, approximately 46.7% of the questions in ORD-QA are derived from multiple document chunks. Furthermore, due to the rigorous and sophisticated dataset creation process, the questions in ORD-QA are significantly more logically complex, requiring both extensive EDA domain knowledge and advanced logical reasoning to solve. This makes ORD-QA highly effective for evaluating the performance of RAG workflows in addressing complex, EDA tool-related queries.

VI. Experimental Results

A. Training Dataset Collection

This section discusses the process of collecting training dataset for the fine-tuning of all models used in RAG-EDA.

Dataset for Text Embedding Model: To construct the EDA-domain-specific {query, positive sample, negative sample} triplets mentioned in Section III-A, we first collected a comprehensive list of 261 EDA terminologies covering areas like logic synthesis, power analysis, partitioning, floorplanning, placement, routing, STA, DRC, and LVS. Our dataset collection process involved selecting a terminology t_i from this list and using GPT-3.5 to generate a related user query (s_i). GPT-3.5 also rephrases the sentence structure of s_i, which is used as the positive sample (s_i^+). Simultaneously, we created a negative sample (s_i^-) by replacing t_i in s_i with a different terminology t_j resulting in a data triplet $\{s_i, s_i^+, s_i^-\}$. Overall, we gathered 3975 query, positive sample, negative sample triplets to fine-tune the text embedding model.

Dataset For Generator: There are two training phases for generator: 1) domain knowledge pretrain and 2) task-specific instruction tuning. For the construction of the pretraining dataset, we divided the two textbooks [42], [43] mentioned in Section III-D into 4863 chunks, ensuring that each chunk does not exceed 1024 tokens. We design a rigorous flow for generating the training corpus for task-specific instruction tuning: Initially, we sample ten document chunks from the OpenROAD documentation to form a document pool. Subsequently, GPT-4 is prompted to select n ($n \leq 10$) document chunks from this pool that are logically connected. Based on the n selected document chunks (r), we prompt GPT-4 to generate one question (q) from the perspective of the OpenROAD users, analyze the question step by step, and give the answer (a) referring to r. Following the procedure above, we generate 1732 $\{q, r, a\}$ triplets as our dataset for instruction tuning. Due to the logical rigorousness of the flow and the extraordinary performance of GPT-4, it is guaranteed that the vast majority of the generated training data is error-free and of high quality. Moreover, we also conducted manual check on the GPT-4-generated training data and has fixed a few logical errors.

Dataset for Reranker Model: To construct the dataset of reranker finetuning, we first randomly sample triplets from the instruction-tuning dataset. For each question q_i, we treat their corresponding relevant document chunks r_i as the positive documents and denote them by C_i^+. Then, we apply hybrid retrieval described in Section III-B on q_i, and set the top-k values for both the lexical and semantic searchers to be 10, in result, 20 document chunks are retrieved and denoted as C_i'. Among the chunks in C_i', we prompt GPT-4 to detect the ones which are weakly related/irrelevant with q_i, and denote them by C_i^-. Finally, we get 208 $\{q_i, C_i^+, C_i^-\}$ triplets as the dataset for reranker finetuning.

To construct the dataset for the LLM reranker, we use the OpenROAD document and the content of two EDA textbooks [42], [43] as the document sources. The dataset construction flow consists of three steps.

1) We randomly select one document chunk from the document sources, referring to which we prompt GPT-4 to generate the corresponding user query q.
2) For q, we apply hybrid retrieval introduced in Section III-B to retrieve 10 documents (denoted as D).
3) We prompt GPT-4 to analyze the logical relevance relationship between q and each document in D, and gives the corresponding relevance score. The score ranges from 0 to 100 and the detailed scoring criterion can be found in https://github.com/lesliepy99/RAG-EDA/blob/main/README.md#scoring-criterion-for-query-document-relevanceGithub. After the third step, the textual relevance analysis (denoted as A) and the scoring results (denoted as S) are generated for query q and retrieved documents D. By applying the above construction flow, we get 22744 $\{q, D, A, S\}$ items in the LLM reranker training dataset. Note that the textual relevance analysis A is exactly the "analysis" part in Fig. 8.

B. Experimental Setting

In our customized RAG flow, the text embedding model was fine-tuned on bge-large-en-v1.5[2] for 2 epochs with a batch size of 16, a maximum sequence length of 128, and a temperature of 0.05. The reranker, based on the bge-reranker-large model[3] and XLM-RoBERTa architecture, was fine-tuned for 30 epochs with a batch size of 4 and a maximum sequence length of 512.

[2]https://huggingface.co/BAAI/bge-large-en-v1.5
[3]https://huggingface.co/BAAI/bge-reranker-large

The generator, leveraging the Qwen1.5-14B-chat model [45], was trained in two stages. For domain knowledge pretraining, the model was trained for 2 epochs on the textbook chunks dataset described in section VI-A with a learning rate of 1e-5 and a batch size of 1. For task-specific instruction tuning, the model was fine-tuned using QLoRA [46] for 4 epochs with a learning rate of 2e-4, a batch size of 1, a maximum sequence length of 4096, and a LoRA rank of 32. The LLM reranker, based on Yi-9B-200K,[4] was fine-tuned in two stages. During the pointwise instruction tuning stage, the model was trained for 1 epoch with a batch size of 4 and a learning rate of 1e-5. In the pairwise LTR training stage, the model was fine-tuned using QLoRA [46] for 1200 steps with a batch size of 4, a learning rate of 2e-4, and a LoRA rank of 16. All the models mentioned above were fine-tuned on a cluster of 16×A100 GPUs, each with 40 GB of memory. For training runtime, the pretraining and instruction tuning phases of the RAG-EDA-generator take around 30 and 75 min, respectively. For the LLM reranker, the pointwise instruction tuning stage and the pairwise LTR stage take around 56 min and 10 h to complete.

C. Evaluation: Text Embedding Model

In this section, we assess the performance of the dense (semantic) search facilitated by our fine-tuned text embedding model, utilizing the ORD-QA benchmark. Initially, we employ the text embedding model currently under evaluation to encode the 290 document chunks, as detailed in Section V, into the vector database. Subsequently, each question q within ORD-QA is encoded by the text embedding model to obtain its vector representation e^q. We then utilize the similarity search library Faiss [57] to identify the top-k closest document chunks to e^q within the vector space. We use the metric recall@k (introduced in Section II-B) to measure the capacity of the text embedding model in retrieving relevant documents. We adopt the text-embedding-ada-002 model by OpenAI, bge-large-en-v1.5, gte-large[5] (embedding model adopted by ORAssistant) and mxbai-embed-large-v1[6] (embedding model adopted by OpenROAD-Assistant) as baselines, and set k to be 5, 10, 15, and 20 for evaluating recall@k. Experimental results in Table I show that fine-tuned by our well-designed contrastive learning scheme on the EDA corpus, our fine-tuned text embedding model dramatically outperforms all baseline models on the task of retrieving relevant documents for EDA-tool-involved questions.

D. Evaluation: Reranker Model

To evaluate the performance of our fine-tuned LLM reranker model on filtering out weakly related and irrelevant documents, for each question q in ORD-QA, hybrid search (described in Section III-B) is first conducted. We use Faiss and BM25 for the lexical and semantic searches, respectively, and set the search limit of each retriever to be 20. After hybrid document search, we obtain 40 candidate relevant document chunks for q. Then, we de-duplicate the candidates. Next, the reranker model calculates the similarity score between q and each candidate document chunk, and the top k document chunks with the highest scores are selected as the relevant document chunks. Subsequently, we can calculate recall@k to measure the fine-filtering ability of the reranker. The open-source reranker model bge-reranker-large, bge-reranker-base[7] (reranker model adopted by ORAssistant), the LLM-based reranker model bge-reranker-v2-gemma,[8] and the SOTA domain customized reranker RAG-EDA reranker [47] are selected as the baselines. We also compare our reranker model with RRF (introduced in Section II) since it is used in RAG-fusion, one commonly used RAG flow. Table II lists the experimental results on ORD-QA: by setting k from 1 to 5, our fine-tuned LLM reranker model outperforms the baselines in filtering out weakly related documents and preserve relevant documents, for EDA-tool-related questions and documents. Especially compared with the SOTA EDA-domain reranker model by RAG-EDA [47], our proposed LLM reranker improves the top-1 to top-5 reranking recall by 10.6%, 5.0%, 4.9%, 4.3%, and 4.9%, respectively.

TABLE I
PERFORMANCE OF SEMANTIC SEARCH FOR TEXT EMBEDDING MODEL ON ORD-QA

Model type	recall@5	recall@10	recall@15	recall@20
text-embedding-ada-002	0.447	0.534	0.609	0.634
bge-large-en-v1.5	0.503	0.596	0.634	0.660
gte-large	0.398	0.509	0.559	0.608
mxbai-embed-large-v1	0.329	0.435	0.491	0.553
Our embedding model	**0.547**	**0.658**	**0.702**	**0.733**

TABLE II
PERFORMANCE OF RERANKER FOR WEAKLY RELATED DOCUMENTS FILTERING ON ORD-QA

Model type	recall@1	recall@2	recall@3	recall@4	recall@5
RRF	0.248	0.342	0.391	0.435	0.484
bge-reranker-large	0.360	0.441	0.484	0.497	0.522
bge-reranker-base	0.354	0.435	0.509	0.584	0.596
bge-reranker-v2-gemma	0.404	0.503	0.559	0.584	0.627
RAG-EDA reranker [47]	0.335	0.534	0.609	0.665	0.671
LLM Reranker (ours)	**0.441**	**0.584**	**0.658**	**0.708**	**0.720**

E. Evaluation: Generator

To assess the performance of our proposed generator on EDA-tool-related questions, each question q in ORD-QA is combined with its golden relevant documents r into a predesigned prompt, which is fed to the generator for answer generation. We evaluate the generator's performance by calculating BLEU, ROUGE-L, and UniEval scores for each generated answer against the ground-truth answer. We choose OpenAI GPT-4 [36], o1-preview [48] and several SOTA open-source chat LLMs [45], [49], [50] as baseline models. Experimental results presented in Table III indicate that our generator, trained using the two-stage EDA-specific training scheme outlined earlier, is equipped with domain

[4]https://huggingface.co/01-ai/Yi-9B-200K
[5]https://huggingface.co/thenlper/gte-large
[6]https://huggingface.co/mixedbread-ai/mxbai-embed-large-v1
[7]https://huggingface.co/BAAI/bge-reranker-base
[8]https://huggingface.co/BAAI/bge-reranker-v2-gemma

TABLE III
PERFORMANCE OF THE LLMS AS GENERATORS ON ORD-QA

Model Type	ORD-QA#functionality			ORD-QA#vlsi-flow			ORD-QA#gui & install & test			ORD-QA#all		
	BLEU	ROUGE-L	UniEval	BLEU	ROUGE-L	UniEval	BLEU	ROUGE-L	UniEval	BLEU	ROUGE-L	UniEval
GPT-4 [36]	0.116	0.257	0.724	0.149	0.268	0.759	0.184	0.328	0.798	0.141	0.277	0.751
o1-preview [48]	0.067	0.179	0.750	0.067	0.172	0.780	0.082	0.216	**0.817**	0.071	0.186	0.774
llama-2-13B-chat [49]	0.099	0.224	0.694	0.110	0.214	0.654	0.125	0.250	0.678	0.108	0.228	0.680
Baichuan2-13B-Chat [50]	0.089	0.244	0.743	0.073	0.260	0.781	0.092	0.281	0.740	0.086	0.257	0.751
Qwen1.5-14B-Chat [45]	0.077	0.189	0.604	0.099	0.175	0.513	0.127	0.272	0.739	0.095	0.206	0.615
Qwen1.5-14B-Chat finetuned w. PPO [40]	0.086	0.210	0.720	0.123	0.235	0.735	0.132	0.278	0.735	0.106	0.233	0.727
Qwen1.5-14B-Chat finetuned w. DPO [41]	0.084	0.207	0.723	0.116	0.220	0.736	0.128	0.262	0.748	0.103	0.223	0.732
RAG-EDA-generator (ours)	**0.150**	**0.319**	**0.788**	**0.188**	**0.326**	**0.798**	**0.224**	**0.374**	0.809	**0.177**	**0.334**	**0.795**

TABLE IV
PERFORMANCE OF THE RAG FLOWS ON ORD-QA

RAG Flow	ORD-QA#functionality			ORD-QA#vlsi-flow			ORD-QA#gui & install & test			ORD-QA#all		
	BLEU	ROUGE-L	UniEval	BLEU	ROUGE-L	UniEval	BLEU	ROUGE-L	UniEval	BLEU	ROUGE-L	UniEval
Vanilla-RAG [34]	0.088	0.204	0.635	0.089	0.206	0.669	0.139	0.258	0.726	0.101	0.217	0.665
RAG-fusion [26], [51]	0.080	0.193	0.624	0.088	0.202	0.681	0.150	0.274	0.698	0.099	0.215	0.656
HyDE [52]	0.080	0.188	0.603	0.075	0.185	0.658	0.130	0.255	0.712	0.091	0.204	0.643
llmlingua [53]–[55]	0.056	0.179	0.565	0.048	0.162	0.586	0.091	0.221	0.658	0.062	0.185	0.593
ITER-RETGEN [56]	0.098	0.208	0.652	0.089	0.202	0.632	0.136	0.251	0.700	0.105	0.217	0.659
ORAssistant [24]	0.046	0.155	0.600	0.060	0.163	0.695	0.065	0.183	0.691	0.054	0.164	0.645
ORAssistant with LLM reranker	0.047	0.154	0.602	0.0597	0.163	0.722	0.0750	0.194	0.672	0.057	0.166	0.648
OpenROAD-Assistant [23]	0.044	0.187	0.643	0.061	0.181	0.742	0.062	0.200	0.671	0.053	0.189	0.674
OpenROAD-Assistant with LLM reranker	0.053	0.213	0.671	0.045	0.189	**0.774**	0.072	0.222	0.751	0.056	0.209	0.716
RAG-EDA+GPT-4	0.101	0.230	0.681	0.116	0.234	0.690	0.178	0.299	0.742	0.123	0.248	0.698
RAG-EDA [47]	**0.119**	**0.281**	0.699	0.147	0.269	0.746	0.166	0.302	0.776	0.137	0.283	0.729
RAG-EDA with LLM reranker	0.117	0.267	**0.727**	**0.164**	**0.283**	0.728	**0.179**	**0.342**	**0.832**	**0.144**	**0.290**	**0.752**

knowledge in EDA. It demonstrates the capability to accurately answer questions related to EDA tool documentation, surpassing both commercial and academic chat LLMs for the overall performance on the ORD-QA benchmark. Notably, due to the complex reasoning nature of o1-preview [48], its performance on ORD-QA tends to produce answers that are longer and more redundant compared to the ground-truth answers, resulting in relatively lower BLEU and ROUGE-L scores. Nevertheless, owing to its strong reasoning capabilities, o1-preview achieves UniEval scores that rank second only to RAG-EDA-generator on the overall ORD-QA benchmark. Furthermore, for the "gui & install & test" question type, o1-preview even surpasses our fine-tuned RAG-EDA-generator in terms of UniEval performance.

To assess the effectiveness of our two-stage training scheme for generator training in EDA tool documentation QA, we compare it with two state-of-the-art RLHF techniques: PPO [40] and DPO [41]. For both methods, we augment the instruction tuning dataset from Section VI-A, using original answers as "chosen" samples and generating lower-quality "rejected" samples via Qwen1.5-14B-chat. Qwen1.5-14B-chat serves as the base model for the policy and reward models in PPO/DPO. All models are fine-tuned for one epoch using LoRA on 4 NVIDIA A100 GPUs (80GB), with a batch size of 8 per device and a maximum sequence length of 4096. For DPO, the learning rate is 2e-4 with a beta value of 0.1, while PPO uses learning rates of 1e-5 for the policy model and 1e-4 for the reward model. As shown in Table III, our two-stage training scheme outperforms PPO and DPO on the ORD-QA benchmark. This is due to the inherent complexity of EDA tool documentation QA, which requires intricate logical reasoning. Such complexity makes it difficult to reliably evaluate answer correctness, limiting the quality of DPO training data and hindering the robustness of PPO's reward model.

F. Evaluation: RAG Flow

Finally, we evaluate the performance of our customized RAG flow. During the implementation of RAG-EDA, we use Faiss and BM25 as the semantic and lexical search engines for the retriever, and set the top-k value of each searcher to 20. The top-k of the reranker is set to 5. For answer generation, our generator model is quantized by 4-bit for efficient inference. We select five cutting-edge academic RAG flows as our baselines, and they mainly differs in the way to retrieve documents. *Vanilla-RAG* [34] solely applies semantic retrieval. *RAG-fusion* [51] uses both semantic and lexical retrievals and applies RRF [26] for documents fusion and reranking. For *HyDE* [52], hypothetical documents are appended to the query for accurate semantic retrieval. *LLMlingua* [53], [54], [55] renders compression for the user query and retrieved documents for key information extraction and efficient inference. Finally, *ITER-RETGEN* [56] synergizes retrieval and generation in an iterative manner. Moreover, we also evaluate OpenROAD-Assistant [23] and ORAssistant [24], two SOTA EDA tool QA RAG frameworks.

For implementation of vanilla-RAG, HyDE, LLMlingua, ITER-RETGEN and RAG-Fusion, we use Faiss and OpenAI's text-embedding-ada-002 model for semantic retrieval, BM25 for lexical retrieval, and GPT-4 as the generator. To ensure fair comparison, the top-k value for semantic retrieval in vanilla-RAG, HyDE, LLMlingua, ITER-RETGEN and OpenROAD-Assistant is set to 5. For ORAssistant, the top-k values of the retriever and reranker are set to 20 and 5, and GPT-4 is used as the generator. For RAG-fusion, semantic and lexical retrievers have top-k values of 20, with RRF selecting the top-5 highest-scoring documents as relevant. For the ablation study, we replace RAG-EDA's generator with GPT-4 as a baseline. As shown in Table IV, ORD-QA questions are grouped into three consolidated categories: 1) functionality;

2) VLSI-flow; and 3) GUI & install & test (the latter combines smaller categories). Columns 2 to 4 present results for these categories, while the final column shows the average across all questions. Performance metrics (BLEU, ROUGE-L, UniEval) compare generated answers from the RAG flow to ground-truth answers in ORD-QA. Our proposed RAG flow (RAG-EDA) achieves the best performance across all categories, with the baseline "RAG-EDA+GPT-4" (replacing the generator with GPT-4) ranking second. This highlights the effectiveness of our RAG flow and fine-tuned generator for EDA tool documentation QA. RAG-fusion, HyDE, ITER-RETGEN, and LLMlingua underperform compared to our customized flow, emphasizing the importance of tailoring the RAG flow for EDA documentation QA tasks. In the QA process of ORAssistant, the LLM is first prompted to identify a subset of document chunks relevant to the user question prior to the retrieval phase. The retriever is then applied to this subset to retrieve the most relevant documents. However, since ORD-QA includes a significant portion of questions that span document chunks across various aspects, this retrieval mechanism often results in incomplete document retrieval, leading to reduced performance. For OpenROAD-Assistant, the primary factor behind the lower QA quality is the relatively low accuracy of document retrieval.

To evaluate the QA generation improvement from higher reranking accuracy, we integrate our proposed LLM reranker into the retrieval phase of RAG-EDA, ORAssistant, and OpenROAD-Assistant. As shown in Table IV, incorporating the LLM reranker improves RAG-EDA's overall BLEU, ROUGE-L, and UniEval scores by 0.7%, 0.7%, and 2.3%, respectively. However, for functionality questions, RAG-EDA with the LLM reranker tends to generate slightly more redundant answers than the ground truth, resulting in lower BLEU and ROUGE-L scores. For some vlsi-flow questions, despite the LLM reranker achieves higher retrieval recall, the ground-truth documents are mostly concentrated in the latter part of the top-5 reranking results. This causes the generator to focus on less relevant documents in the earlier part, leading to a lower UniEval score. For OpenROAD-Assistant, integrating the LLM reranker enhances overall ORD-QA performance and achieves the highest UniEval score for VLSI-flow questions among all evaluated RAG flows. Meanwhile, due to its two-round query mechanism, ORAssistant restricts retrieval to a subset of OpenROAD document chunks, which can result in incomplete retrieval for questions requiring multiple ground-truth documents in ORD-QA. In such cases, the integration of the LLM reranker has limited impact on retrieval accuracy, resulting in QA performance similar to that of ORAssistant.

Fig. 10 presents the runtime analysis of RAG-EDA and baseline RAG flows on the ORD-QA benchmark. Llmlingua achieves the shortest runtime due to its use of compressed prompts, which reduce generator inference latency. Vanilla-RAG follows with the second shortest runtime, thanks to its simple retrieval process. RAG-EDA, RAG-EDA+GPT-4, and RAG-fusion show similar runtimes, with GPT-4 slightly faster than the finetuned generator. HyDE and ITER-RETGEN take longer runtimes: HyDE increases generator load by appending a hypothetical document to the query, while ITER-RETGEN's iterative approach raises computational costs. Moreover, OpenROAD-Assistant and ORAssistant have the longest runtimes. ORAssistant's two-round query mechanism is time-consuming, while OpenROAD-Assistant's use of a finetuned adapter slows the inference speed compared to a merged model. Finally, incorporating the LLM reranker into RAG-EDA increases the average runtime per QA by only 1.78 s. It is noteworthy that, compared to RAG [47] and RAG-EDA with the LLM reranker, RAG-EDA+GPT-4 achieves comparable performance while requiring less runtime. The primary reason for its faster inference lies in GPT-4 being a production-grade LLM, deployed on the cloud with access to more powerful hardware resources and extensive optimizations in both algorithms and hardware utilization. In contrast, while our locally deployed models (RAG-EDA-generator and the LLM reranker) have slower inference speeds, they offer significant advantage in terms of cost-efficiency. Additionally, they mitigate concerns related to private data leakage, making them a more reliable and practical choice in the scenario of EDA tool documentation QA.

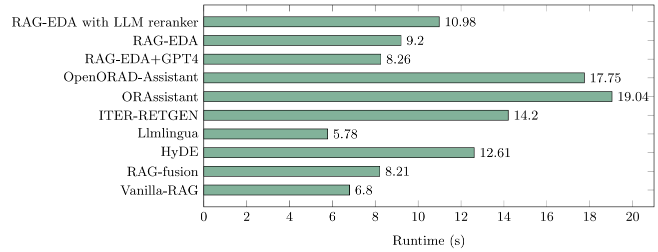

Fig. 10. Average runtime of the RAG flows to process one question in ORD-QA.

The dataset adheres to the same standards as ORD-QA and comprises 60 question-document-answer triplets. Similar to ORD-QA, the questions are categorized into three types: 1) functionality; 2) vlsi-flow; and 3) gui accounting for 48.3%, 36.7%, and 15.0% of the dataset, respectively. It is worth noting that the question types installation and test are not included, as such topics are not covered in the commercial EDA tool documentation. Among these, 47 questions are from one single document chunk, while 11 and 2 questions require two and three document chunks, respectively. To prevent the disclosure of proprietary API information from commercial EDA tools when using OpenAI GPT-4, all API names in the QA dataset and documentation are anonymized using six-letter random strings prior to experimentation. The results obtained using the anonymized dataset are presented in Table V. The experimental results presented in Tables VI and VII demonstrate the effectiveness of our fine-tuned embedding model and LLM-based reranker in document retrieval for the commercial-tool QA dataset. Specifically, our LLM reranker achieves the highest recall at top-1 through top-4, and ranks second only to the RAG-EDA reranker [47] at recall@5. As shown in Table V, the RAG-EDA pipeline enhanced with the LLM reranker outperforms all SOTA baselines in terms of overall BLEU and ROUGE-L scores. For the UniEval metric, it ranks second, following only RAG-EDA [47].

TABLE V
PERFORMANCE OF THE RAG FLOWS ON QA DATASET OF THE COMMERCIAL EDA TOOL. WE DENOTE THE COMMERCIAL TOOL QA DATASET AS COMM-QA FOR SHORT

RAG Flow	COMM-QA#functionality			COMM-QA#vlsi-flow			COMM-QA#gui			COMM-QA#all		
	BLEU	ROUGE-L	UniEval	BLEU	ROUGE-L	UniEval	BLEU	ROUGE-L	UniEval	BLEU	ROUGE-L	UniEval
Vanilla-RAG [34]	0.077	0.183	0.549	0.075	0.187	0.576	0.051	0.168	0.617	0.072	0.182	0.569
RAG-Fusion [26], [51]	0.169	0.280	0.697	0.152	0.272	0.655	0.133	0.269	0.572	0.157	0.275	0.663
HyDE [52]	0.144	0.263	0.663	0.118	0.252	0.634	0.124	0.257	0.601	0.131	0.258	0.643
llmlingua [53]–[55]	0.082	0.217	0.570	0.071	0.208	0.523	0.079	0.234	0.570	0.078	0.216	0.553
ITER-RETGEN [56]	0.172	0.290	0.674	0.157	0.274	0.682	0.152	0.284	0.668	0.163	0.283	0.676
RAG-EDA+GPT-4	**0.180**	0.293	0.621	0.173	0.274	0.618	**0.194**	0.318	0.646	0.179	0.290	0.624
RAG-EDA [47]	0.173	0.311	**0.755**	0.179	0.306	0.688	0.182	**0.359**	**0.752**	0.176	0.332	**0.730**
RAG-EDA with LLM reranker	0.176	**0.331**	0.693	**0.194**	**0.352**	**0.691**	0.158	0.349	0.698	**0.180**	**0.341**	0.693

TABLE VI
PERFORMANCE OF SEMANTIC SEARCH FOR TEXT EMBEDDING MODELS ON QA DATASET OF THE COMMERCIAL EDA TOOL

Model type	recall@5	recall@10	recall@15	recall@20
text-embedding-ada-002	0.480	0.560	0.587	0.680
bge-large-en-v1.5	0.547	0.613	0.653	0.707
gte-large	0.547	0.600	0.640	0.707
mxbai-embed-large-v1	0.547	0.600	0.653	0.707
Our embedding model	**0.773**	**0.827**	**0.827**	**0.867**

TABLE VII
PERFORMANCE OF RERANKER FOR WEAKLY RELATED DOCUMENTS FILTERING ON ORD-QA

Model type	recall@1	recall@2	recall@3	recall@4	recall@5
RRF	0.573	0.653	0.707	0.747	0.760
bge-reranker-large	0.573	0.667	0.707	0.747	0.787
bge-reranker-base	0.453	0.627	0.653	0.680	0.707
bge-reranker-v2-gemma	0.587	0.707	0.747	0.747	0.787
RAG-EDA reranker [47]	0.573	0.667	0.707	0.747	**0.867**
LLM Reranker (ours)	**0.693**	**0.747**	**0.773**	**0.827**	0.840

TABLE VIII
PERFORMANCE OF THE RAG FLOWS ON EDA-CORPUS QA BENCHMARK

RAG Flow	BERTScore			LLM Evaluation		
	Precision	Recall	F1	True	Partial	False
ORAssistant	0.816	0.872	0.842	128	4	64
ORAssistant w. LLM reranker	0.815	0.876	0.844	146	5	45
OpenROAD-Assistant	0.846	0.871	0.857	123	20	53
OpenROAD-Assistant w. LLM Reranker	0.835	0.878	0.854	137	7	52
RAG-EDA	0.824	0.886	0.853	157	17	22
RAG-EDA w. LLM Reranker	0.823	0.886	0.853	162	11	23

Further analysis reveals that RAG-EDA with the LLM reranker achieves the best performance on vlsi-flow questions across all three metrics. Meanwhile for the functionality and gui categories, the original RAG-EDA (without the LLM reranker) performs better, which can be attributed to its higher reranking recall on these specific question types. In conclusion, the experimental findings validate that the proposed RAG-EDA framework, when combined with an LLM-based reranker, can be effectively adapted for documentation-based QA across a range of EDA tools.

G. Evaluation on EDA-Corpus

In this section, we evaluate our proposed framework on the EDA-Corpus nonaugmented QA benchmark, which consists of 196 QA pairs, and compare it with two state-of-the-art EDA QA systems: OpenROAD-Assistant [23] and ORAssistant [24]. To facilitate the evaluation, we annotate the ground-truth document chunk(s) for each QA pair based on the OpenROAD documentation. According to the annotation results, 164 (83.7%) questions in the EDA-Corpus can be fully answered using a single document chunk from the OpenROAD documentation. In contrast, the remaining 32 (16.3%) questions do not have any relevant document chunk because they either stem from out-of-scope GitHub discussions, reference outdated APIs, or involve general EDA knowledge not documented in OpenROAD. The complete annotation file is available at Github.[9] The evaluation focuses on the document retrieval phase, including both the retriever and reranker models. For the retriever evaluation, we compare the performance of the RAG-EDA-embedding model, ORAssistant-retriever (gte-large), and OpenROAD-Assistant-retriever (mxbai-embed-large-v1). For the reranker evaluation, since OpenROAD-Assistant does not utilize a reranker, we evaluate the ORAssistant-reranker (bge-reranker-base), RAG-EDA-reranker, and our proposed LLM reranker. The evaluations of both the retriever and reranker models follow the experimental setups described in Sections VI-C and VI-D. The experimental results, visualized in Fig. 11(a) and (b), highlight the effectiveness of our proposed retriever and reranker models in the document retrieval task for the EDA-Corpus nonaugmented QA benchmark.

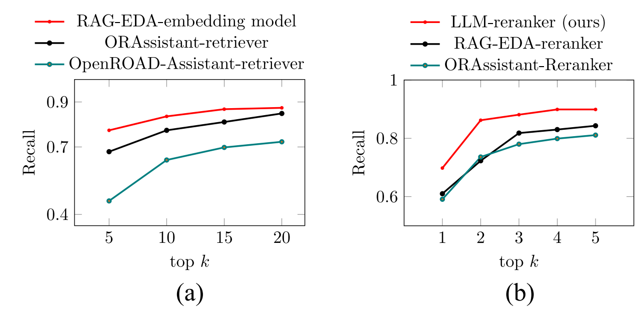

Fig. 11. Evaluation of the (a) retriever and (b) reranker models for document retrieval, on the EDA-Corpus QA benchmark.

For the evaluation of the complete RAG flow, RAG-EDA and ORAssistant follow the experimental setup described in Section VI-F. For OpenROAD-Assistant, due to the absence of a reranker model, only the retriever model is used during the document retrieval phase, with the retrieval top-k set to 5. For performance evaluation, we adopt the same metrics

[9]https://github.com/lesliepy99/RAG-EDA/blob/main/benchmark/EDA-Corpus-document-annotation.xlsx

as OpenROAD-Assistant [23] and EDA-Corpus [22], namely, BERTScore [58] and a GPT-4-based LLM evaluation score. For the LLM evaluation, given a question q, a ground-truth answer gt, and a generated answer a, GPT-4 is prompted to classify a into the following categories: *True* (if a contains factual information and correctly answers q), *Partially True* (if a answers q correctly but includes incorrect or irrelevant information), and *False* (if a fails to answer q or contains false information). The experimental results, summarized in Table VIII, show that among ORAssistant, OpenROAD-Assistant, and RAG-EDA, the BERTScores are comparable. However, based on the LLM evaluation, RAG-EDA [47] achieves the best performance, with the highest number of "True" classifications and the fewest "False" classifications. To further assess the effectiveness of the LLM reranker within the RAG flow, we integrate the LLM reranker into the retriever phase of all three RAG systems. As shown in Table VIII, incorporating the LLM reranker generally enhances the answer quality, as reflected in improved LLM evaluation scores.

VII. Conclusion

We propose RAG-EDA, a customized RAG flow for EDA tool documentation QA. We develop domain-specific training strategies and use EDA-related corpus to customize the text embedding, reranker and generator models for RAG-EDA. Moreover, we propose to use LLM as the reranker model in RAG-EDA and customize a two-stage training scheme for the LLM reranker. Finally, we develop and release ORD-QA, a documentation QA evaluation benchmark based on OpenROAD. RAG-EDA with the LLM reranker model has achieved much better performance on both ORD-QA and a commercial EDA tool documentation QA benchmark compared with other state-of-the-arts.

Acknowledgment

The authors are grateful to Shutong Lin and Hairuo Han from The Chinese University of Hong Kong, for their assistance with experimental setup.

References

[1] T. Ajayi et al., "Toward an open-source digital flow: First learnings from the OpenRoad project," in *Proc. DAC*, 2019, pp. 1–4.
[2] X. Li et al., "iEDA: An open-source intelligent physical implementation toolkit and library," 2023, *arXiv:2308.01857*.
[3] Z. He et al., "ChatEDA: A large language model powered autonomous agent for EDA," in *Proc. MLCAD*, 2023, pp. 1–6.
[4] H. Wu et al., "ChatEDA: A large language model powered autonomous agent for EDA," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 43, no. 10, pp. 3184–3197, Oct. 2024.
[5] M. Liu et al., "ChipNeMo: Domain-adapted LLMs for chip design," 2023, *arXiv:2311.00176*.
[6] K. Chang et al., "ChipGPT: How far are we from natural language hardware design," 2023, *arXiv:2305.14019v3*.
[7] M. Liu, N. Pinckney, B. Khailany, and H. Ren, "Verilogeval: Evaluating large language models for Verilog code generation," in *Proc. ICCAD*, 2023, pp. 1–8.
[8] Y. Fu et al., "Gpt4aigchip: Towards next-generation AI accelerator design automation via large language models," in *Proc. ICCAD*, 2023, pp. 1–9.
[9] J. Blocklove, S. Garg, R. Karri, and H. Pearce, "Chip-Chat: Challenges and opportunities in conversational hardware design," in *Proc. MLCAD*, 2023, pp. 1–6.
[10] S. Thakur, J. Blocklove, H. Pearce, B. Tan, S. Garg, and R. Karri, "AutoChip: Automating HDL generation using LLM feedback," *arXiv:2311.04887*, 2023.
[11] Y. Lu, S. Liu, Q. Zhang, and Z. Xie, "RTLLM: An open-source benchmark for design RTL generation with large language model," in *Proc. ASPDAC*, 2024, pp. 722–727.
[12] S. Thakur et al., "Benchmarking large language models for automated Verilog RTL code generation," in *Proc. DATE*, 2023, pp. 1–6.
[13] S. Thakur et al., "VeriGen: A large language model for Verilog code generation," *ACM Trans. Design Autom. Electron. Syst.*, vol. 29, no. 3, p. 46, 2023.
[14] Z. Pei, H.-L. Zhen, M. Yuan, Y. Huang, and B. Yu, "BetterV: Controlled Verilog generation with discriminative guidance," 2024, *arXiv:2402.03375*.
[15] Y. Tsai, M. Liu, and H. Ren, "RTLFixer: Automatically fixing RTL syntax errors with large language models," 2023, *arXiv:2311.16543*.
[16] M. Orenes-Vera, A. Manocha, D. Wentzlaff, and M. Martonosi, "AutoSVA: Democratizing formal verification of RTL module interactions," in *Proc. DAC*, 2021, pp. 535–540.
[17] R. Kande et al., "LLM-assisted generation of hardware assertions," 2023, *arXiv:2306.14027*.
[18] X. Meng et al., "Unlocking hardware security assurance: The potential of LLMs," 2023, *arXiv:2308.11042*.
[19] S. Paria, A. Dasgupta, and S. Bhunia, "DIVAS: An LLM-based end-to-end framework for SoC security analysis and policy-based protection," 2023, *arXiv:2308.06932v1*.
[20] B. Ahmad, S. Thakur, B. Tan, R. Karri, and H. Pearce, "Fixing hardware security bugs with large language models," 2023, *arXiv:2302.01215*.
[21] X. Yao et al., "HDLdebugger: Streamlining HDL debugging with large language models," 2024, *arXiv:2403.11671*.
[22] B.-Y. Wu et al., "EDA corpus: A large language model dataset for enhanced interaction with OpenROAD," 2024, *arXiv:2405.06676*.
[23] U. Sharma, B.-Y. Wu, S. R. D. Kankipati, V. A. Chhabria, and A. Rovinski, "OpenRoad-assistant: An open-source large language model for physical design tasks," in *Proc. MLCAD*, 2024, pp. 1–7.
[24] A. Kaintura, S. S. Luar, I. I. Almeida, and R. Palaniappan, "ORAssistant: A custom RAG-based conversational assistant for OpenROAD," 2024, *arXiv:2410.03845*.
[25] V. Karpukhin et al., "Dense passage retrieval for open-domain question answering," 2020, *arXiv:2004.04906*.
[26] G. V. Cormack, C. L. Clarke, and S. Buettcher, "Reciprocal rank fusion outperforms condorcet and individual rank learning methods," in *Proc. SIGIR*, 2009, pp. 758–759.
[27] J. Chen, S. Xiao, P. Zhang, K. Luo, D. Lian, and Z. Liu, "BGE M3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation," 2024, *arXiv:2402.03216*.
[28] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "BLEU: A method for automatic evaluation of machine translation," in *Proc. ACL*, 2002, pp. 311–318.
[29] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Proc. Workshop Text Summarization Branches Out*, 2004, pp. 74–81.
[30] M. Zhong et al., "Towards a unified multi-dimensional evaluator for text generation," in *Proc. EMNLP*, 2022, pp. 2023–2038.
[31] S. Xiao, Z. Liu, P. Zhang, and N. Muennighof, "C-Pack: Packaged resources to advance general Chinese embedding," 2023, *arXiv:2309.07597*.
[32] T. Gao, X. Yao, and D. Chen, "SimCSE: Simple contrastive learning of sentence embeddings," 2021, *arXiv:2104.08821*.
[33] J. Zhang, Z. Lan, and J. He, "Contrastive learning of sentence embeddings from scratch," 2023, *arXiv:2305.15077*.
[34] P. Lewis et al., "Retrieval-augmented generation for knowledge-intensive NLP tasks," *Proc. NIPS*, vol. 33, 2020, pp. 9459–9474.
[35] F. Cuconasu et al., "The power of noise: Redefining retrieval for RAG systems," 2024, *arXiv:2401.14887*.
[36] J. Achiam et al., "GPT-4 technical report," 2023, *arXiv:2303.08774*.
[37] A. Kumar et al., "Training language models to self-correct via reinforcement learning," 2024, *arXiv:2409.12917*.
[38] S. Welleck et al., "Generating sequences by learning to self-correct," 2022, *arXiv:2211.00053*.
[39] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.
[40] L. Ouyang et al., "Training language models to follow instructions with human feedback," in *Proc. NIPS*, 2022, pp. 1–68.
[41] R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn, "Direct preference optimization: Your language model is secretly a reward model," in *Proc. NIPS*, vol. 36, 2024, pp. 1–14.

[42] L. Lavagno, I. L. Markov, G. Martin, and L. K. Scheffer, *Electronic Design Automation For IC Implementation, Circuit Design, and Process Technology: Circuit Design, and Process Technology*. Boca Raton, FL, USA: CRC Press, 2016.
[43] C. J. Alpert, D. P. Mehta, and S. S. Sapatnekar, *Handbook of Algorithms for Physical Design Automation*. Boca Raton, FL, USA: CRC Press, 2008.
[44] C. Burges et al., "Learning to rank using gradient descent," in *Proc. ICML*, 2005, pp. 89–96.
[45] J. Bai et al., "Qwen technical report," 2023, *arXiv:2309.16609*.
[46] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient finetuning of quantized LLMs," in *Proc. NIPS*, 2024, pp. 1–26.
[47] Y. Pu, Z. He, T. Qiu, H. Wu, and B. Yu, "Customized retrieval augmented generation and benchmarking for EDA tool documentation QA," 2024, *arXiv:2407.15353*.
[48] *Learning to Reason With Language Models*, OpenAI Res., San Francisco, CA, USA, 2024.
[49] H. Touvron et al., "Llama 2: Open foundation and fine-tuned chat models," 2023, *arXiv:2307.09288*.
[50] A. Yang et al., "Baichuan 2: Open large-scale language models," 2023, *arXiv:2309.10305*.
[51] S. Bruch, S. Gai, and A. Ingber, "An analysis of fusion functions for hybrid retrieval," *ACM Trans. Inf. Syst.*, vol. 42, no. 1, pp. 1–35, 2023.
[52] L. Gao, X. Ma, J. Lin, and J. Callan, "Precise zero-shot dense retrieval without relevance labels," 2022, *arXiv:2212.10496*.
[53] H. Jiang, Q. Wu, C.-Y. Lin, Y. Yang, and L. Qiu, "LLMLingua: Compressing prompts for accelerated inference of large language models," in *Proc. EMNLP*, 2023, pp. 1–19.
[54] H. Jiang et al., "LongLLMLingua: Accelerating and enhancing LLMs in long context scenarios via prompt compression," 2023, *arXiv:2310.06839*.
[55] Z. Pan et al., "LLMLingua-2: Data distillation for efficient and faithful task-agnostic prompt compression," 2024, *arXiv:2403.12968*.
[56] Z. Shao, Y. Gong, Y. Shen, M. Huang, N. Duan, and W. Chen, "Enhancing retrieval-augmented large language models with iterative retrieval-generation synergy," 2023, *arXiv:2305.15294*.
[57] M. Douze et al., "The Faiss library," 2024, *arXiv:2401.08281*.
[58] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi, "BERTScore: Evaluating text generation with bert," 2019, *arXiv:1904.09675*.

Yuan Pu received the B.S. degree in computer science from The Chinese University of Hong Kong, Hong Kong, in 2022, where he is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering, under the supervision of Prof. B. Yu.

His research interest includes combinatorial algorithms/AI in EDA.

Zhuolun He received the B.S. degree in computer science and engineering from Peking University, Beijing, China, in 2017, and the Ph.D. degree from the Chinese University of Hong Kong, Hong Kong, in 2023.

He is currently a Postdoctoral Fellow with the Department of Computer Science and Engineering, Chinese University of Hong Kong.

Yuqi Jiang received the master's degree in new generation electronic information technology from Beijing Jiaotong University, Beijing, China, in 2023. She is currently pursuing the Ph.D. degree with the College of Information Science and Electronic Engineering, Zhejiang University, Hangzhou, China.

Her research interests include multimodal large models and image processing.

Tairu Qiu received the M.Eng. degree in computer engineering from Fudan University, Shanghai, China.

He is currently working with ChatEDA Lab, Foshan, China, where focus on the integration of large language models with electronic design automation. His research interests include generative AI and large language models.

Haoyuan Wu received the B.Eng. degree in software engineering from Sun Yat-sen University, Guangzhou, China, in 2022, and the M.Sc. degree in computer science from The Chinese University of Hong Kong, Hong Kong, in 2023, where he is currently pursuing the Ph.D. degree with the Department of Computer Science and Engineering.

His research interests include generative AI, efficient deep learning, and design automation.

Qi Sun (Member, IEEE) received the doctorate degree from the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong, in 2022.

He then engaged in Postdoctoral Research with Cornell University, Ithaca, NY, USA. He is currently a ZJU100 Young Professor with Zhejiang University, Hangzhou, China. His research areas include AI for EDA, large language model, design technology co-optimization, and RISC-V SoC.

Dr. Sun has been awarded two ICCAD Best Paper Awards and a DATE Best Paper Nomination Award.

Cheng Zhuo (Senior Member, IEEE) received the B.S. and M.S. degrees from Zhejiang University, Hangzhou, China, in 2005 and 2007, respectively, and the Ph.D. degree from the University of Michigan at Ann Arbor, Ann Arbor, MI, USA, in 2010.

He is currently a Qiushi Distinguished Professor with Zhejiang University, Hangzhou, China. He has authored/co-authored more than 200 technical papers. His research focuses on hardware intelligence, machine learning-assisted EDA, and low power designs.

Dr. Zhuo has received five Best Paper Awards and six Best Paper Nominations. He is the recipient of the ACM/SIGDA Meritorious Service Award and Technical Leadership Award, JSPS Faculty Invitation Fellowship, and Humboldt Research Fellowship. He has served on the organization/technical program committees of many international conferences, as the Area Editor for *Journal of Computer Aided Design & Computer Graphics*, and as an Associate Editor for IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, *ACM Transactions on Design Automation of Electronic Systems*, and *Integration* (Elsevier). He is IEEE CEDA Distinguished Lecturer and a Fellow of IET.

Bei Yu (Senior Member, IEEE) received the Ph.D. degree from The University of Texas at Austin, Yx, USA, in 2014.

He is currently an Associate Professor with the Department of Computer Science and Engineering, The Chinese University of Hong Kong, Hong Kong.

Dr. Yu received 11 Best Paper Awards from ICCAD 2024, 2021, and 2013, IEEE TSM 2022, DATE 2022, ASPDAC 2021 and 2012, ICTAI 2019, Integration, the VLSI Journal in 2018, ISPD 2017, SPIE Advanced Lithography Conference 2016, six ICCAD/ISPD contest awards, and many other awards, including the DAC Under-40 Innovator Award in 2024, the IEEE CEDA Ernest S. Kuh Early Career Award in 2022, and the Hong Kong RGC Research Fellowship Scheme (RFS) Award in 2024. He has served as the TPC Chair of ACM/IEEE Workshop on Machine Learning for CAD, and in many journal editorial boards and conference committees.