Customized Retrieval Augmented Generation and Benchmarking for EDA Tool Documentation QA

Yuan Pu[1,2], Zhuolun He[1,2], Tairu Qiu[2], Haoyuan Wu[3], Bei Yu[1]
[1]The Chinese University of Hong Kong, Hong Kong SAR
[2]ChatEDA Tech, China
[3]Shanghai AI Laboratory, Shanghai, China

ABSTRACT

Retrieval augmented generation (RAG) enhances the accuracy and reliability of generative AI models by sourcing factual information from external databases, which is extensively employed in document-grounded question-answering (QA) tasks. Off-the-shelf RAG flows are well pretrained on general-purpose documents, yet they encounter significant challenges when being applied to knowledge-intensive vertical domains, such as electronic design automation (EDA). This paper 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 LLM, and a generative LLM fine-tuned with high-quality domain corpus. Furthermore, 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. The ORD-QA benchmark and the training dataset for our customized RAG flow are open-source at https://github.com/lesliepy99/RAG-EDA.

1 INTRODUCTION

Electronic design automation (EDA) comprises a set of software tools dedicated for the design, analysis and verification of electronic systems. To meet the increasing demand of semiconductor manufacturing and accommodate the advancing technology nodes, current EDA tools support a complex design flow with various functionalities and commands at all levels. Open-source EDA tools such as OpenROAD[1] and iEDA[2] provide complete RTL-to-GDSII flow and environment for circuit design. Commercial software, with rigorous consideration of industrial scenarios, supports much more sophisticated and various functionalities. 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 improve user experience and development efficiency of tool users, application engineers are employed and trained by vendors to provide on-site customer service and support. Nevertheless, the high training and manpower cost has pushed people to consider an automated, no-human-in-the-loop means for such support.

With the exceptional performance and rapid advancement of large language models (LLMs), retrieval augmented generation (RAG) has recently been proven to be an effective method for document grounded QA. As is shown in Figure 1, to prepare for RAG,

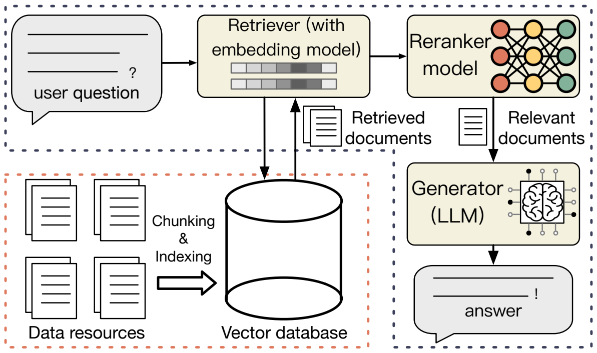


Figure 1: Illustration of the RAG flow.

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

During the process of RAG, the robust reasoning and inference capabilities of LLMs are effectively utilized, while the retrieved relevant information mitigates the hallucination issue and improves the reliability of LLM generation. However, due to the mutually entangled structures of EDA tool documentations, sophisticated nature of EDA flow and rigorous logic reasoning involved in EDA-tool-related questions, applying existing RAG flows for EDA tool documentation QA presents significant challenges. The primary challenge is that existing RAG components lack expert knowledge in EDA, leading to inferior performance in EDA-specific contexts: First, for retriever, existing text embedding models demonstrate poor semantic understanding of the terminologies and concepts of EDA, and are prone to retrieve irrelevant documents by similarity matching. Second, existing reranker models fail to accurately differentiate between relevant and weakly-related documents under EDA-tool-specific scenarios. For example, while the question is "what is the command to place a specific pin", the weakly-related document, which introduces the command to place all pins and is not helpful to answer the question, will be recognized as relevant by existing reranker models. Third, existing generators (chat LLMs) lack EDA domain knowledge and demonstrate limited logical inference capabilities when addressing EDA-related questions, leading to inadequate answer generation. Although many previous research work have customized LLMs for several EDA tasks including script generation [30, 31], HDL generation [32–41], code verification and analysis [42–48], to the best of our knowledge, there has not been any research conducted on the documentation QA of EDA tools.

To resolve the challenges mentioned above, we propose **RAG-EDA**, a retrieval augmented generation (RAG) flow customized for EDA tool documentation question-answering. Our approach consists of three strategic enhancements: (1) For retriever, we first create a corpus of queries and answers that incorporate EDA domain knowledge and terminologies. Using this corpus, we employ contrastive learning to finetune a text embedding model, 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.

Furthermore, 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 evaluate the robustness, universality and transferability of our proposed flow, we also design an internal QA evaluation dataset on the documentation of a commercial EDA tool.

Our major contributions are summarized as follows:

- 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.
- 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.
- We propose a two-stage training scheme, namely, domain-knowledge pre-train and task-specific instruction tuning, to finetune an open-source chat LLM as the generator.
- 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.

2 PRELIMINARIES

2.1 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. Generally, there are two categories of methodology for information retrieval, namely, lexical (sparse) retrieval and semantic (dense) retrieval.

Lexical (sparse) Retrieval. In lexical (sparse) retrieval, the user's query is first tokenized into keywords. Techniques such as term frequency-inverse document frequency (TF-IDF) and BM25 are then used to assess the relevance of each keyword to each document chunk in the collection. The top-k document chunks with the highest relevance scores are selected as the most relevant documents.

Semantic (dense) Retrieval. Given a user question and a list of document chunks, the approach of dense retrieval (semantic retrieval) utilizes the pretrained text embedding model to project the question and document chunks into the semantic vector space. The top k document chunks which are the closest to the question in the vector space (thus have the most similar semantics with the question) are selected as the relevant documents [6].

Reciprocal Rank Fusion. Reciprocal rank fusion (RRF) [14] is a method used for data fusion in information retrieval systems. This technique combines the results from multiple retrieval systems (for example, lexical and semantic retrieval) to produce a single, enhanced ranking. Given a set of documents D and a set of document rankings R (each $r \in R$ ranks the relevance of D for a specific retrieval system, and r is essentially a permutation on $1 \ldots |D|$), the RRF score of one document d can be calculated as:

$$\mathrm{RRF}(d \in D) = \sum_{r \in R} \frac{1}{k + r(d)}, \tag{1}$$

where k is one constant and is usually set to 60.

2.2 Performance Measurement

To evaluate the effectiveness of a retriever/reranker in terms of its ability to retrieve all relevant documents/filter out irrelevant documents, we use recall as the metric. Mathematically, recall@k is defined as the proportion of relevant documents that are retrieved among the top k results returned by the retriever/reranker. Let R be the set of all relevant documents in the corpus, and let R_k be the set of relevant documents that appear in the top k results returned by the retriever/reranker. Then, recall@k is formulated as:

$$\text{recall@}k = \frac{|R_k|}{|R|}. \tag{2}$$

To evaluate the performance of a generator (a chat LLM) in answering EDA-tool questions, we use three most popular metrics applied in the scenario of text generation and question-answering, namely, BLEU [3], ROUGE-L [4], and UniEval [5]. These three metrics are used to measure the coherence, conciseness and factual consistency between the generated answer and ground truth answer for each question.

BLEU. Bilingual evaluation understudy (BLEU) is originally designed for measuring machine translation, but currently is widely used for the measurement of text generation. BLEU computes the precision of n-grams in the candidate answer by determining the proportion of n-grams that appear in the reference answer. This is adjusted by a brevity penalty to discourage overly short translations. The metric is calculated as:

$$\mathrm{BLEU} = BP \cdot \exp\left(\sum_{n=1}^{N} w_n \log p_n\right), \tag{3}$$

which can be best understood as the geometric mean of multiple n-gram precision of various sizes. Here, BP (Brevity Penalty) is a multiplicative factor used to penalize short candidate answer, and w_n are weights assigned to each logarithmic n-gram precision (we

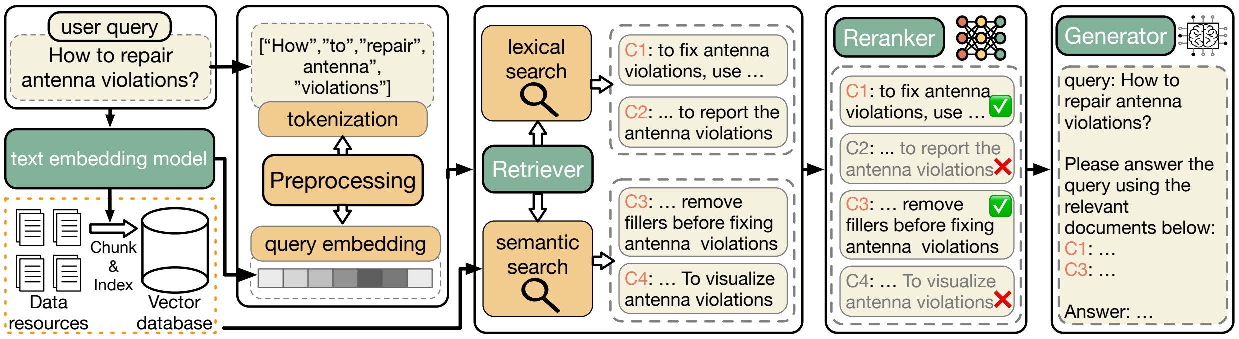

user query
How to repair antenna violations?
text embedding model
Chunk & Index
Data resources
Vector database
["How","to","repair", antenna", "violations"]
tokenization
Preprocessing
query embedding
lexical search
Retriever
semantic search
C1: to fix antenna violations, use …
C2: … to report the antenna violations
C3: … remove fillers before fixing antenna violations
C4: … To visualize antenna violations
Reranker
C1: to fix antenna violations, use …
C2: … to report the antenna violations
C3: … remove fillers before fixing antenna violations
C4: … To visualize antenna violations
Generator
query: How to repair antenna violations?
Please answer the query using the relevant documents below:
C1: …
C3: …
Answer: …

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

use uniform weights). N is the maximal n-gram order considered and 4 is often a practical option.

ROUGE-L. ROUGE-L (recall-oriented understudy for gisting evaluation - longest common subsequence) measures the similarity between the answer generated by the LLM (denoted as X) and the reference answer (denoted as Y) by examining the longest common subsequence (LCS). It computes both recall and precision based on the LCS, and it combines these into an F-measure. The formulas are defined as follows:

$$\begin{aligned} \text{recall}_{\text{LCS}} &= \frac{|\text{LCS}(X,Y)|}{|Y|}, \text{precision}_{\text{LCS}} = \frac{|\text{LCS}(X,Y)|}{|X|}, \\ \text{ROUGE-L} &= \frac{(1+\beta^2)\cdot(\text{precision}_{\text{LCS}}\cdot\text{recall}_{\text{LCS}})}{(\beta^2\cdot\text{precision}_{\text{LCS}})+\text{recall}_{\text{LCS}}}, \end{aligned} \tag{4}$$

where β is typically set to 1.

UniEval. UniEval [5] is a widely-used LLM-based evaluation framework designed to assess the quality of generated text across multiple dimensions, including factual consistency, relevance, coherence, etc. Under the context of EDA tool documentation QA, we use the component of factual consistency score (ranging from 0 to 1, the higher the better) to measure the quality of the generated answer.

3 ALGORITHMS

Figure 2 illustrates the overall flow of **RAG-EDA**, our customized retrieval augmented generation (RAG) flow for EDA tool documentation QA. In the preparation phase, the entire EDA tool documentation is segmented into reasonably sized chunks, which are then encoded into vectors by a customized text embedding model (introduced in Section 3.1), forming a vector database. The first stage is pre-processing, where the user query is tokenized into words and simultaneously encoded into an embedding. The second stage, retrieval (introduced in Section 3.2), operates in two parts: lexical retrieval (TF-IDF or BM25) and semantic retrieval using the pre-built vector database. The results from both searches are combined to form relevant document candidates. Subsequently, the finetuned reranker model (introduced in Section 3.3) conducts fine filtering on the document candidates and further eliminates the weakly-related ones. Finally, the fine-filtered relevant document chunks along with the query are fed to the generator model (introduced in Section 3.4) for answer generation. In the following sub-sections, we will detail the customization and technique for each stage.

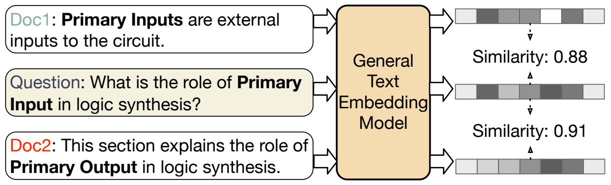
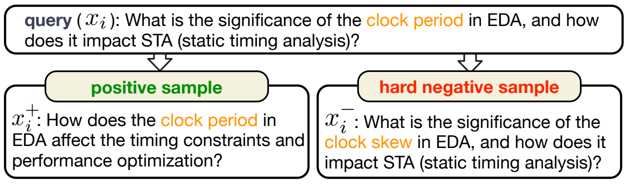

Doc1: Primary Inputs are external inputs to the circuit.
Question: What is the role of Primary Input in logic synthesis?
Doc2: This section explains the role of Primary Output in logic synthesis.
General Text Embedding Model
Similarity: 0.88
Similarity: 0.91

Figure 3: A fail case of general text embedding model in EDA-specific information retrieval (about *Primary Input*). Doc1/Doc2 defines PI/PO. The general text embedding model mistakenly perceives higher similarity between the question and Doc2.

3.1 Domain-Customized Text Embedding

During semantic retrieval, the text embedding model plays a crucial role in transforming both the user query and document chunks into numerical vectors that encapsulate their semantic content. The accuracy of this semantic representation is essential as it directly impacts the retrieval performance. Although many proprietary and open-source text embedding models [7, 8] perform well in general information retrieval tasks, they demonstrate poor semantic understanding of the terminologies and concepts in EDA. As is illustrated in Figure 3, the SOTA general-task text embedding model fails to adequately distinguish between basic EDA terms such as Primary Input (PI) and Primary Output (PO). This deficiency leads to inaccuracies in the semantic vector space it constructs, potentially causing the model to erroneously select less relevant documents. For instance, despite Doc1 being more relevant to the query, Doc2 may be selected as the relevant document candidate due to its perceived semantic closeness, thereby compromising the overall effectiveness of the QA flow.

To address the aforementioned issue and enhance the performance of the retriever, we employ supervised contrastive learning to improve the model's ability to accurately perceive EDA terminologies and concepts. The fundamental principle of our contrastive 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



query (x_i): What is the significance of the clock period in EDA, and how does it impact STA (static timing analysis)?

x_i^+: How does the clock period in EDA affect the timing constraints and performance optimization?

x_i^-: What is the significance of the clock skew in EDA, and how does it impact STA (static timing analysis)?

Figure 4: A contrastive data sample used for text embedding model finetuning.

x_i^- has a similar sentence structure with the query, but pertains to different EDA terminology/concept. Figure 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 [9, 10] 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:

$$-\log \frac{e^{sim(x_i,x_i^+)/\tau}}{\sum_{j=1}^{M}(e^{sim(x_i,x_j^+)/\tau} + e^{sim(x_i,x_j^-)/\tau})}, \qquad (5)$$

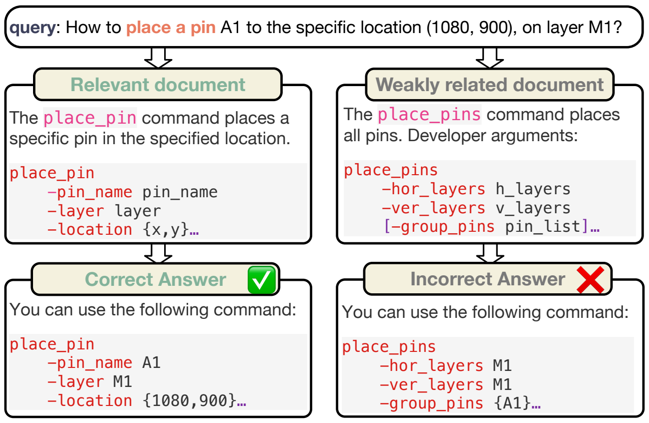
where sim stands for the cosine similarity function, τ is the temperature hyper-parameter.

The text embedding model, finetuned through our supervised contrastive learning approach, exhibits improved semantic comprehension of the EDA-related corpus. This enhancement leads to superior document retrieval performance in the EDA-tool documentation QA task. Detailed results and analysis will be presented in Section 5.

3.2 Hybrid Information Retrieval

For EDA tool documentation QA, user queries span various categories including VLSI flow inquiries, command/option usage, installation guidelines, GUI usage and test unit, each characterized by distinct syntactic structure and sentence composition. Lexical (sparse) retrieval methods, such as TFIDF or BM25, are particularly effective when queries contain specific keywords or phrases directly matching the documentation content. For example, a query about the usage of a specific command like *remove_filler* is efficiently handled by these traditional lexical search techniques, which pinpoint the exact document section defining that command. Conversely, when a query pertains to VLSI flow and does not necessarily contain direct keywords or phrases found in the documentation, semantic (dense) search method is employed. The semantic search focuses on matching the semantics between the query and the document chunks, thereby enhancing retrieval accuracy over lexical searches. For instance, a query like "What are the steps for routing" would enable a semantic search to retrieve document chunks that explain the commands *global_route* and *detail_route*.

Motivated by the preceding analysis, our approach deviates from the conventional methodology of solely employing semantic search for information retrieval [11]. In our proposed framework, we implement a hybrid search strategy, integrating both semantic and lexical search techniques for the retrieval of documents. Document



query: How to place a pin A1 to the specific location (1080, 900), on layer M1?

The place_pin command places a specific pin in the specified location.

```
place_pin
    -pin_name pin_name
    -layer layer
    -location {x,y}…
```

The place_pins command places all pins. Developer arguments:

```
place_pins
    -hor_layers h_layers
    -ver_layers v_layers
    [-group_pins pin_list]…
```

You can use the following command:

```
place_pin
    -pin_name A1
    -layer M1
    -location {1080,900}…
```

You can use the following command:

```
place_pins
    -hor_layers M1
    -ver_layers M1
    -group_pins {A1}…
```

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

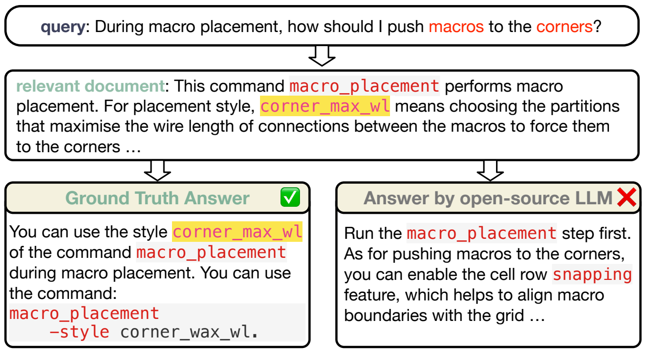
chunks retrieved via the hybrid strategy are merged and deduplicated to form a pool of relevant document candidates. Subsequently, candidate documents that are weakly related to the query will be further differentiated and excluded, as detailed in Section 3.3.

3.3 Reranker Finetuning

Candidate document chunks obtained by the hybrid retriever either share semantic similarities or contain the same keywords/phrases as the query. Based on the relevance to the query, each candidate document chunk can be classified into one of two categories: (1) Relevant document whose content can be directly used to answer the query; (2) Weakly-related document which may contain the same EDA terminology/concept as the query, but can not be used to answer the query. Previous research [12] indicates that weakly-related documents may cause augmented LLMs to generate error-prone responses. Figure 5 demonstrates the impact of a weakly-related document chunk on answer generation for EDA tool queries. The user inquires the command to place a specific pin. If the relevant document of the command *place_pin* is referred to, the generator gets correct answer. Meanwhile, referring to the weakly-related document, that is, the command to place all pins, the generator leads to wrong answer.

Re-ranking is one common technique to filter out weakly-related and irrelevant documents, and is implemented by many frameworks such as LlamaIndex, LangChain and HayStack. SOTA rerankers in the RAG framework typically employ a cross-encoder architecture, which processes the query and each candidate document as a pair. It calculates a relevance score for each pair using a cross-attention mechanism. The top k documents with the highest relevance scores are selected as the relevant documents. Although SOTA cross-encoder reranker models such as bge-reranker[1] demonstrate superior performance in general tasks of information retrieval, they are less effective in distinguishing between relevant and weakly-related documents in the context of EDA tool documentation QA. As is shown in Figure 5, for a query requesting a command to place a specific pin, the reranker incorrectly identifies the command *place_pins* as relevant. We observe that with well-designed prompt, the proprietary LLMs such as GPT-4 perform well

[1]https://huggingface.co/BAAI/bge-reranker-large


query: During macro placement, how should I push macros to the corners?

relevant document: This command `macro_placement` performs macro placement. For placement style, `corner_max_wl` means choosing the partitions that maximise the wire length of connections between the macros to force them to the corners …

Ground Truth Answer ✓

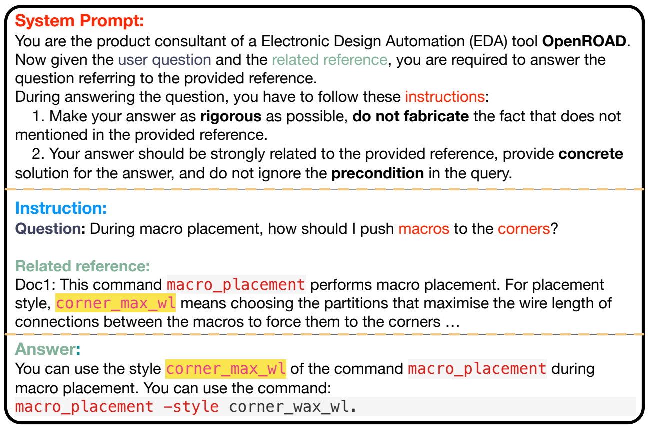
You can use the style `corner_max_wl` of the command `macro_placement` during macro placement. You can use the command:

```
macro_placement
    -style corner_wax_wl.
```

Answer by open-source LLM ✗

Run the `macro_placement` step first. As for pushing macros to the corners, you can enable the cell row `snapping` feature, which helps to align macro boundaries with the grid …

Figure 6: A fail case of existing open-source chat LLMs for answering the EDA-tool questions.


System Prompt:
You are the product consultant of a Electronic Design Automation (EDA) tool **OpenROAD**. Now given the user question and the related reference, you are required to answer the question referring to the provided reference.
During answering the question, you have to follow these instructions:
1. Make your answer as **rigorous** as possible, **do not fabricate** the fact that does not mentioned in the provided reference.
2. Your answer should be strongly related to the provided reference, provide **concrete** solution for the answer, and do not ignore the **precondition** in the query.

Instruction:
Question: During macro placement, how should I push macros to the corners?

Related reference:
Doc1: This command `macro_placement` performs macro placement. For placement style, `corner_max_wl` means choosing the partitions that maximise the wire length of connections between the macros to force them to the corners …

Answer:
You can use the style `corner_max_wl` of the command `macro_placement` during macro placement. You can use the command:

```
macro_placement -style corner_wax_wl.
```

Figure 7: One sample of the QA data instance used from instruction tuning.

in filtering out weakly-related documents for EDA-tool-involved QA . However, the autoregressive nature of GPT-4 makes it slow for document filtering, and proprietary LLMs are also costly and challenging to deploy.

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 $Q = \{q_1, q_2, ..., q_n\}$. For each $q_i \in Q$, we apply the hybrid retriever described in Section 3.2 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 enhance the ability of the reranker model in distinguishing between relevant documents and weakly-related documents, even when they share similar semantics or identical keywords with the query, we introduce a contrastive learning scheme to fine-tune the reranker model using the documents filtered 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 $\{s_{i,1}^-, s_{i,2}^-, ..., s_{i,m}^-\}$ from C_i^- ($m \leq k$), the contrastive loss for q_i can be then written as:

$$-\log \frac{e^{f(q_i, s_i^+)/\tau}}{e^{f(q_i, s_i^+)/\tau} + \sum_{j=1}^{m} e^{f(q_i, s_{i,j}^-)/\tau}}, \tag{6}$$

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.

After being fine-tuned on the high-quality dataset, our reranker model exhibits high retrieval recall in EDA-tool QA scenarios. Detailed experiments and their results are discussed in Section 5.

3.4 Domain-Specific LLM Generator

Serving as the last stage of the retrieval augmented generation (RAG) flow, the generator, usually an LLM, treats the user query and the filtered relevant document(s) as input, and output the answer. We observe that directly applying the open-source chat LLMs as generators for EDA tool documentation QA leads to inferior quality of the answers. Figure 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.

By analysis, we get the conclusion that there are mainly two limitations of existing chat LLMs for EDA tool documentation QA: 1). Open-source chat LLMs lack expert knowledge in EDA. Questions involving the usage of EDA tools are usually complicated, besides the provided relevant document, some extra domain knowledge in EDA may be helpful in answering the question. 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 Figure 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.

To guarantee better quality of answer generation, we propose a two-stage training scheme to finetune the chat LLM as the generator. The first stage is domain knowledge pre-train: We prepare two well-known textbooks in EDA, namely, *Electronic Design Automation for IC Implementation, Circuit Design, and Process Technology* [20] and *Handbook of Algorithms for Physical Automation* [21], as pre-train corpus. We divide the two textbooks into chunks by paragraph and construct a dataset, on which we pre-train the chat LLM in the autoregressive manner. 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 groud-truth answer. Then, we integrate a well-designed system prompt to each triplet to form a QA dataset. Figure 7 shows one QA sample in the dataset. Finally, we finetune the pre-trained chat LLM in stage 1 on the collected QA dataset.

Note that large language models (LLM) follow the autoregressive manner, which predicts tokens sequentially and the prediction of each token depends on the tokens that have been previously generated. Given an LLM with parameter θ and a token sequence $x = \{x_1, x_2, ..., x_n\}$, the probability of the LLM to predict x is formulated as:

$$P(x) = \sum_{t=1}^{n} P(x_t | x_{<t}; \theta). \tag{7}$$

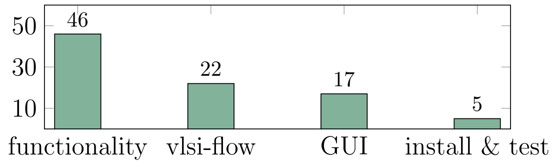


Figure 8: Distribution of question categories in ORD-QA.

The objective of LLM training is to minimized the negative log-likelihood loss formulated in Equation (8):

$$L = -\sum_{t=1}^{n} \log P(x_t | x_{<t}; \theta). \tag{8}$$

The two stages, namely, domain-knowledge pre-train and task-specific intrudction tuning, are trained using the loss function defined in Equation (8), on their respective corpus dataset.

4 BENCHMARK

To systematically and quantitatively evaluate the performance of RAG-EDA, we created and published the **ORD-QA** dataset, which is open-source at https://github.com/lesliepy99/RAG-EDA. ORD-QA contains 90 high-quality question-document-answer triplets based on the OpenROAD [1] documentation. For developing the dataset, we started by segmenting the complete OpenROAD documentation[2] into 290 chunks, based on the markdowns of sections and subsections. Following the segmentation, each time we randomly sampled 1 to 5 document chunk(s). If only one document chunk is sampled or there is logical relationship among the sampled chunks, we prompted GPT-4 to formulate a EDA-related question based on the document chunk(s), and analyzed the question alongside the provided document chunks to generate corresponding answer. Each generated question-document-answer triplet underwent a rigorous manual review to correct any logical inconsistencies and fallacies. Additionally, we manually crafted realistic user queries about the OpenROAD documentation and provided accurate answers to further enhance the dataset. The final ORD-QA dataset contains 90 high-quality <question, reference documents, answer> triplets.

The questions in the ORD-QA dataset are categorized into four types: functionality, VLSI-flow, GUI, and installation & test. Functionality questions detail the use of specific commands or options in particular user scenarios. VLSI-flow questions request the steps to complete a given VLSI process. GUI questions address the usage of the OpenROAD GUI, while installation & test questions cover software installation guidelines and unit test module inquiries. Figure 8 shows the number distribution of different question categories in ORD-QA. Most of the questions (82.2%) are formulated based on 1 or 2 document chunks. The rest (17.8%) are based on more than 3 document chunks. Among the 90 questions, 89 of them refer to no more than 5 document chunks, with only one question based on 7 document chunks.

5 EXPERIMENTAL RESULTS

5.1 Training Dataset Collection

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

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

Dataset for Text Embedding Model. To construct the EDA-domain-specific <query, positive sample, negative sample> triplets mentioned in Section 3.1, 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 produced the answer for (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 3,975 <query, positive sample, negative sample> triplets to fine-tune the text embedding model.

Dataset For Generator. There are two training phases for generator: domain knowledge pre-train and task-specific instruction tuning. For the construction of pre-train dataset, we split the two textbooks [20, 21] mentioned in Section 3.4 into 4863 chunks, where the token size of each chunk is limited within 1024. The procedure of dataset collecting for task-specific instruction tuning is as follows: Each time we sample 10 document chunks of OpenROAD documentation as the document pool. Then, GPT-4 is prompted to select n ($n \leq 10$) document chunks from the pool such that the selected chunks are logically related. 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.

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 3.2 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.

5.2 Experimental Setting

In our customized RAG flow, the text embedding model was fine-tuned on bge-large-en-v1.5[3] 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[4] with the XLM-RoBERTa architecture [22], underwent fine-tuning for 30 epochs, with batch size 4 and maximum sequence length of 512. The generator, powered by the Qwen-14b-chat model [23], was trained in two stages: For domain knowledge pre-train, we pretrained the model on the textbook chunks dataset described in section 5.1 for 2 epochs with the learning rate of 1e-5 and batch size of 1. For task-specific instruction tuning, we finetuned the model using QLoRA[28] for 4 epochs with the learning rate of 2e-4 and batch

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

Table 1: 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
Our embedding model	**0.547**	**0.658**	**0.702**	**0.733**

Table 2: 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
Our reranker	0.335	**0.534**	**0.609**	**0.665**	**0.671**

size of 1. The maximal sequence length is 4096 and the lora rank is 32. All the models above were fine-tuned on 16xA100 with 40G memory each.

5.3 Evaluation: Text Embedding Model

In this sub-section, we evaluate the performance of dense (semantic) search by our finetuned text embedding model, on the benchmark ORD-QA. We first use the under-test text embedding model to encode the 290 document chunks in Section 4 into the vector database. Then, each question q in ORD-QA is encoded by the text embedding model to obtain its vector representation e^q, and the similarity search library Faiss [29] is used to search for the top-k closest document chunks with e^q in the vector space. We use the metric recall@k (introduced in Section 2.2) to measure the capacity of the text embedding model in retrieving relevant documents. We adopt the text-embedding-ada-002 model by OpenAI and bge-large-en-v1.5 as baselines, and set k to be 5, 10, 15 and 20 for evaluating recall@k. Experimental results in Table 1 show that fine-tuned by our well-designed constrastive learning scheme on the EDA corpus, our fine-tuned text embedding model dramatically outperforms its base model (bge-large-en-v1.5) and the SOTA commercial text embedding model by OpenAI, on the task of retrieving relevant documents for EDA-tool-involved questions.

5.4 Evaluation: Reranker Model

To evaluate the performance of our fine-tuned reranker model on filtering out weakly-related and irrelevant documents, for each question q in ORD-QA, hybrid search (described in Section 3.2) 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 SOTA open-source reranker model bge-reranker-large, based on which our reranker is fine-tuned, is selected as one baseline. We also compare our reranker model with RRF (introduced in Section 2) since it is used in RAG-fusion, one commonly-used RAG flow. Table 2 lists the experimental results on ORD-QA: by setting k from 1 to 5, our fine-tuned reranker model outperforms the baselines in filtering out weakly-related documents and preserve relevant documents, for EDA-tool-related questions and documents.

5.5 Evaluation: Generator

To evaluate the performance of our pre-trained and fine-tuned generator on the task of answering EDA-tool-involved questions, for each question q in ORD-QA, we combine q and its golden relevant documents r into a pre-designed prompt (Figure 7 shows the template of the prompt). The prompt is then fed to the generator for answer generation. To quantitatively measure the performance of the generator, for each question, the BLEU, ROUGE-L and UniEval scores between the generated answer and the ground-truth answer are calculated. We select OpenAI GPT-4 and several SOTA open-source chat LLMs as our baselines, and experimental results in Table 3 show that being trained by our proposed two-stage EDA-specific training scheme described in Section 3.4, our generator is equipped with domain knowledge in EDA and possesses the ability of answering questions related to EDA tool documentation, and outperforms both commercial and academic chat LLMs on the ORD-QA benchmark.

5.6 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 5 cutting-edge academic RAG flows as our baselines, and they mainly differs in the way to retrieve documents. **Vanilla-RAG** [11] solely applies semantic retrieval. **RAG-fusion** [13] uses both semantic and lexical retrievals and applies RRF [14] for documents fusion and re-ranking. For **HyDE** [15], hypothetical documents are appended to the query for accurate semantic retrieval. **LLMlingua** [16–18] renders compression for the user query and retrieved documents for key information extraction and efficient inference. Finally, **ITER-RETGEN** [19] synergizes retrieval and generation in an iterative manner.

For the implementation of the baselines, we use Faiss and OpenAI 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 values of semantic retrieval for vanilla-RAG, HyDE, Llmlingua and ITER-RETGEN are set to 5. For RAG-fusion, the top-k values of semantic and lexical retrievers are set to 20, and RRF returns the top-5 highest-score documents as relevant. The number of retrieve-generation iterations of ITER-RETGEN is set to 2. For ablation study, we also replace the generator of RAG-EDA by GPT-4 as one baseline. Table 4 shows the experimental results on ORD-QA: The questions in ORD-QA can be divided into four categories as mentioned in Section 4, and since the numbers of GUI questions and install & test questions are small, we combine them together as a new category. Consequently, columns 2 to 4 in Table 4 list results for the three categories of questions, and the last column shows the average results on the whole benchmark. Note that the metrics of BLEU, ROUGE-L and UniEval are calculated

Table 3: 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 [27]	0.116	0.257	0.724	0.149	0.268	0.759	0.184	0.328	0.798	0.141	0.277	0.751
Qwen1.5-14B-Chat [23]	0.077	0.189	0.604	0.099	0.175	0.513	0.127	0.272	0.739	0.095	0.206	0.615
llama-2-13B-chat [24]	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 [25]	0.089	0.244	0.743	0.073	0.260	0.781	0.092	0.281	0.740	0.086	0.257	0.751
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 4: 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 [11]	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 [13, 14]	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 [15]	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 [16–18]	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 [19]	0.098	0.208	0.652	0.089	0.202	0.632	0.136	0.251	0.700	0.105	0.217	0.659
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 (ours)	**0.119**	**0.281**	**0.699**	**0.147**	**0.269**	**0.746**	**0.166**	**0.302**	**0.776**	**0.137**	**0.283**	**0.729**

Table 5: Performance of the RAG flows on QA dataset of the commercial EDA tool.

RAG Flow	BLEU	ROUGE-L	UniEval
Vanilla-RAG [11]	0.072	0.182	0.569
RAG-fusion [13, 14]	0.157	0.275	0.663
HyDE [15]	0.131	0.258	0.643
llmlingua [16–18]	0.078	0.216	0.553
ITER-RETGEN [19]	0.163	0.283	0.676
RAG-EDA+GPT-4	**0.179**	0.290	0.624
RAG-EDA (ours)	0.176	**0.332**	**0.730**

between the ground-truth answers (provided in ORD-QA) and the generated answers by the RAG flow. It is observed that compared with the question categories of functionality and vlsi-flow, the GUI & install & test questions are easier to solve in nature, on which all the RAG flows achieve better performance. Our proposed RAG flow (RAG-EDA) achieves the best performance on all categories of questions, and the baseline "RAG-EDA+GPT-4" (replacing the generator with GPT-4 in our proposed flow) ranks second, demonstrating effectiveness of our proposed RAG flow and fine-tuned generator in solving documentation QA for EDA tools. The performances of RAG-fusion, HyDE and ITER-RETGEN are similar and worse than our customized flow, indicating the necessity of RAG flow customization for EDA tool documentation QA. The result of Llmlingua is the worst, the reason is that the compression LLM of Llmlingua demonstrates inferior performance on handling EDA-tool related questions and documents. After the process of prompt compression, the key information in both the question and the documents are missing, leading to bad quality for answering.

Figure 9 shows the runtime analysis of RAG-EDA and the baseline RAG flows on the ORD-QA benchmark. Llmlingua achieves the shortest average runtime, attributed to its use of compressed prompts which reduces the inference latency of the generator. Vanilla-RAG has the second shortest runtime due to its straightforward retrieval process. The runtimes of RAG-EDA, RAG-EDA+GPT4 and RAG-fusion are similar, and the inference runtime of GPT-4 is slightly faster than our finetuned generator. HyDE and ITER-RETGEN take the longest runtime. HyDE increases the generator's inference load by appending a hypothetical document to the query as a prompt, whereas ITER-RETGEN operates in an iterative manner, leading to higher computational costs.

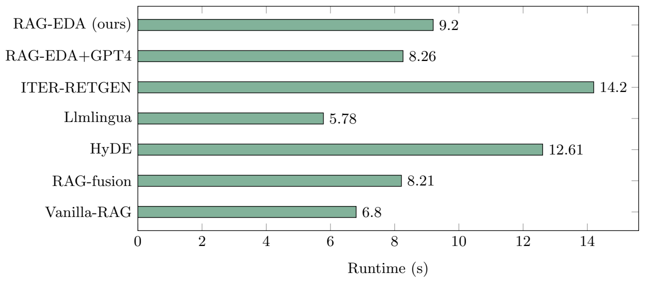


Figure 9: The average runtime of the RAG flows to process one question in ORD-QA.

To further verify the transferability and universality of our proposed RAG flow for different EDA tools, we collect a documentation QA evaluation dataset for a commercial EDA tool (the tool is a platform for timing ECO). The dataset follows the same standard as ORD-QA and contains 60 question-documents-answer triplets. Experimental results in Table 5 shows the outperformance of our proposed RAG flow compared with the baselines, thus certifies that our proposed RAG flow can be transferred to the task of documentation QA for different EDA tools.

6 CONCLUSION

We propose RAG-EDA, a customized retrieval augmented generation (RAG) flow for electronic design automation (EDA) tool documentation question-answering (QA). We develop domain-specific training strategies and use EDA-related corpus to customize the text embedding, reranker and generator models for RAG-EDA. Besides, we also develop and release ORD-QA, a documentation QA evaluation benchmark based on OpenROAD. RAG-EDA has achieved much better performance on both ORD-QA and a commercial EDA tool documentation QA benchmark compared with other state-of-the-arts.

REFERENCES

[1] T. Ajayi, V. A. Chhabria, M. Fogaça, S. Hashemi, A. Hosny, A. B. Kahng, M. Kim, J. Lee, U. Mallappa, M. Neseem *et al.*, "Toward an open-source digital flow: First learnings from the openroad project," 2019, pp. 1–4.

[2] X. Li, S. Tao, Z. Huang, S. Chen, Z. Zeng, L. Ni, Z. Huang, C. Zhuang, H. Wu, W. Li *et al.*, "ieda: An open-source intelligent physical implementation toolkit and library," 2023.

[3] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: a method for automatic evaluation of machine translation."

[4] C.-Y. Lin, "ROUGE: A Package for Automatic Evaluation of Summaries," in *Workshop on Text Summarization Branches Out*, 2004, pp. 74–81.

[5] M. Zhong, Y. Liu, D. Yin, Y. Mao, Y. Jiao, P. Liu, C. Zhu, H. Ji, and J. Han, "Towards a Unified Multi-Dimensional Evaluator for Text Generation," 2022, pp. 2023–2038.

[6] V. Karpukhin, B. Oğuz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W.-t. Yih, "Dense passage retrieval for open-domain question answering," 2020.

[7] S. Xiao, Z. Liu, P. Zhang, and N. Muennighof, "C-pack: Packaged resources to advance general chinese embedding," 2023.

[8] 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.

[9] T. Gao, X. Yao, and D. Chen, "Simcse: Simple contrastive learning of sentence embeddings," 2021.

[10] J. Zhang, Z. Lan, and J. He, "Contrastive learning of sentence embeddings from scratch," 2023.

[11] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel *et al.*, "Retrieval-augmented generation for knowledge-intensive nlp tasks," vol. 33, pp. 9459–9474, 2020.

[12] F. Cuconasu, G. Trappolini, F. Siciliano, S. Filice, C. Campagnano, Y. Maarek, N. Tonellotto, and F. Silvestri, "The power of noise: Redefining retrieval for rag systems," 2024.

[13] S. Bruch, S. Gai, and A. Ingber, "An analysis of fusion functions for hybrid retrieval," *ACM Transactions on Information Systems*, vol. 42, no. 1, pp. 1–35, 2023.

[14] G. V. Cormack, C. L. Clarke, and S. Buettcher, "Reciprocal rank fusion outperforms condorcet and individual rank learning methods," 2009, pp. 758–759.

[15] L. Gao, X. Ma, J. Lin, and J. Callan, "Precise zero-shot dense retrieval without relevance labels," 2022.

[16] H. Jiang, Q. Wu, C.-Y. Lin, Y. Yang, and L. Qiu, "LLMLingua: Compressing prompts for accelerated inference of large language models," 2023.

[17] H. Jiang, Q. Wu, , X. Luo, D. Li, C.-Y. Lin, Y. Yang, and L. Qiu, "LongLLMLingua: Accelerating and enhancing llms in long context scenarios via prompt compression," 2023.

[18] Z. Pan, Q. Wu, H. Jiang, M. Xia, X. Luo, J. Zhang, Q. Lin, V. Ruhle, Y. Yang, C.-Y. Lin, H. V. Zhao, L. Qiu, and D. Zhang, "LLMLingua-2: Data distillation for efficient and faithful task-agnostic prompt compression," 2024.

[19] 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.

[20] 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.* CRC Press, 2016.

[21] C. J. Alpert, D. P. Mehta, and S. S. Sapatnekar, *Handbook of algorithms for physical design automation.* CRC press, 2008.

[22] A. Conneau, K. Khandelwal, N. Goyal, V. Chaudhary, G. Wenzek, F. Guzmán, E. Grave, M. Ott, L. Zettlemoyer, and V. Stoyanov, "Unsupervised cross-lingual representation learning at scale," 2019.

[23] J. Bai, S. Bai, Y. Chu, Z. Cui, K. Dang, X. Deng, Y. Fan, W. Ge, Y. Han, F. Huang *et al.*, "Qwen technical report," 2023.

[24] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," 2023.

[25] A. Yang, B. Xiao, B. Wang, B. Zhang, C. Bian, C. Yin, C. Lv, D. Pan, D. Wang, D. Yan *et al.*, "Baichuan 2: Open large-scale language models," 2023.

[26] Z. Cai, M. Cao, H. Chen, K. Chen, K. Chen, X. Chen, X. Chen, Z. Chen, Z. Chen, P. Chu *et al.*, "Internlm2 technical report," 2024.

[27] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," 2023.

[28] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," 2024.

[29] M. Douze, A. Guzhva, C. Deng, J. Johnson, G. Szilvasy, P.-E. Mazaré, M. Lomeli, L. Hosseini, and H. Jégou, "The faiss library," 2024.

[30] Z. He, H. Wu, X. Zhang, X. Yao, S. Zheng, H. Zheng, and B. Yu, "Chateda: A large language model powered autonomous agent for eda," 2023, pp. 1–6.

[31] H. Wu, Z. He, X. Zhang, X. Yao, S. Zheng, H. Zheng, and B. Yu, "Chateda: A large language model powered autonomous agent for eda," 2024.

[32] M. Liu, T.-D. Ene, R. Kirby, C. Cheng, N. Pinckney, R. Liang, J. Alben, H. Anand, S. Banerjee, I. Bayraktaroglu *et al.*, "Chipnemo: Domain-adapted llms for chip design," 2023.

[33] K. Chang, Y. Wang, H. Ren, M. Wang, S. Liang, Y. Han, H. Li, and X. Li, "Chipgpt: How far are we from natural language hardware design," 2023.

[34] M. Liu, N. Pinckney, B. Khailany, and H. Ren, "Verilogeval: Evaluating large language models for verilog code generation." IEEE, 2023, pp. 1–8.

[35] Y. Fu, Y. Zhang, Z. Yu, S. Li, Z. Ye, C. Li, C. Wan, and Y. C. Lin, "Gpt4aigchip: Towards next-generation ai accelerator design automation via large language models." IEEE, 2023, pp. 1–9.

[36] J. Blocklove, S. Garg, R. Karri, and H. Pearce, "Chip-chat: Challenges and opportunities in conversational hardware design." IEEE, 2023, pp. 1–6.

[37] S. Thakur, J. Blocklove, H. Pearce, B. Tan, S. Garg, and R. Karri, "Autochip: Automating hdl generation using llm feedback," 2023.

[38] Y. Lu, S. Liu, Q. Zhang, and Z. Xie, "Rtllm: An open-source benchmark for design rtl generation with large language model." IEEE, 2024, pp. 722–727.

[39] S. Thakur, B. Ahmad, Z. Fan, H. Pearce, B. Tan, R. Karri, B. Dolan-Gavitt, and S. Garg, "Benchmarking large language models for automated verilog rtl code generation." IEEE, 2023, pp. 1–6.

[40] S. Thakur, B. Ahmad, H. Pearce, B. Tan, B. Dolan-Gavitt, R. Karri, and S. Garg, "Verigen: A large language model for verilog code generation," 2023.

[41] Z. Pei, H.-L. Zhen, M. Yuan, Y. Huang, and B. Yu, "BetterV: Controlled Verilog Generation with Discriminative Guidance," 2024.

[42] Y. Tsai, M. Liu, and H. Ren, "Rtlfixer: Automatically fixing rtl syntax errors with large language models," 2023.

[43] M. Orenes-Vera, A. Manocha, D. Wentzlaff, and M. Martonosi, "Autosva: Democratizing formal verification of rtl module interactions." IEEE, 2021, pp. 535–540.

[44] R. Kande, H. Pearce, B. Tan, B. Dolan-Gavitt, S. Thakur, R. Karri, and J. Rajendran, "Llm-assisted generation of hardware assertions," 2023.

[45] X. Meng, A. Srivastava, A. Arunachalam, A. Ray, P. H. Silva, R. Psiakis, Y. Makris, and K. Basu, "Unlocking hardware security assurance: The potential of llms," 2023.

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

[47] B. Ahmad, S. Thakur, B. Tan, R. Karri, and H. Pearce, "Fixing hardware security bugs with large language models," 2023.

[48] X. Yao, H. Li, T. H. Chan, W. Xiao, M. Yuan, Y. Huang, L. Chen, and B. Yu, "Hdldebugger: Streamlining hdl debugging with large language models," 2024.