Invited: Infusing EDA Knowledge into LLM Systems: An Information-Source Perspective

Yuhan Qin
The Chinese University of Hong Kong
Hong Kong SAR

Yuan Pu
The Chinese University of Hong Kong
Hong Kong SAR

Tairu Qiu
ChatEDA Tech
China

Zhuolun He
The Chinese University of Hong Kong
Hong Kong SAR

Bei Yu
The Chinese University of Hong Kong
Hong Kong SAR

Abstract

Large language models have shown remarkable potential for electronic design automation (EDA), yet building effective LLM systems for EDA remains challenging due to complex tool-specific terminology and documentation. This paper surveys knowledge injection techniques that infuse domain expertise into LLM systems for EDA. We examine three complementary approaches: finetuning, which encodes EDA knowledge into model parameters through training on domain corpora and synthetic data; retrieval-augmented generation (RAG), which dynamically retrieves from external knowledge bases; and multi-agent flow, which decomposes complex tasks across specialized agents and leverages environment feedback for iterative refinement. As a case study, we present a graph-based RAG approach that addresses global queries requiring cross-chunk reasoning. The method trains document-customized embeddings via contrastive learning on knowledge graphs, detects semantically related entities using HDBSCAN clustering, and generates textual summaries integrated through hybrid retrieval. Experiments on OpenROAD documentation demonstrate significant improvements in answering global queries while maintaining local query performance. These findings highlight that domain customization is essential for effective knowledge injection, and graph-based techniques are particularly promising as they inherently encode domain knowledge through entity extraction and relationship modeling.

CCS Concepts

• **Hardware → Electronic design automation**.

Keywords

LLM4EDA

ACM Reference Format:

Yuhan Qin, Yuan Pu, Tairu Qiu, Zhuolun He, and Bei Yu. 2026. Invited: Infusing EDA Knowledge into LLM Systems: An Information-Source Perspective. In *Proceedings of the 2026 International Symposium on Physical Design (ISPD '26), March 15–18, 2026, Bonn, Germany*. ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3764386.3779640

1 Introduction

Large language models (LLMs) have demonstrated remarkable capabilities in natural language understanding and code generation, enabling intelligent automation across diverse domains. Their ability to process complex instructions, generate human-like text, and reason over diverse contexts has naturally drawn attention from the electronic design automation (EDA) community, where automation and knowledge management are critical challenges [1–3].

Recent years have witnessed an explosion of LLM applications across various EDA tasks. For RTL and HDL design, [4–10] target code generation from natural language specifications; [11, 12] address high-level synthesis and accelerator design; [13] optimizes existing RTL code; [14–16] focus on debugging and error correction. For hardware verification and security, [17, 18] automate formal property specification; [19–21] target vulnerability analysis. For EDA tool assistance, [22–25] enable documentation question answering; [26–29] support script generation; [30, 31] facilitate interactive tool usage. For analog and mixed-signal design, [32–34] target circuit sizing and topology generation; [35, 36] automate analog layout. For manufacturing and DFM, [37, 38] enable wafer defect analysis; [39–41] optimize lithography processes. For specification and documentation, [42] generates and reviews design specifications; [43] assist documentation tasks. For SoC and system-level design, [44] provides IC design assistance; [45] enables design space exploration; [46] automates backend design flows.

However, applying LLMs effectively to EDA tasks presents significant challenges stemming from the highly specialized nature of EDA domains. General-purpose LLMs lack adequate understanding of EDA-specific terminology, tool command syntax, and design workflow conventions. EDA documentation uses domain-specific vocabulary that differ from general language usage, while each tool has unique command formats and scripting idioms. Without sufficient domain knowledge, LLMs tend to hallucinate incorrect commands, violate tool conventions, or generate syntactically valid but functionally incorrect outputs.

To address these challenges, researchers have developed systematic approaches for infusing EDA-specific knowledge into LLM systems, as illustrated in Figure 1. Three primary techniques have emerged, each injecting domain knowledge through distinct mechanisms. **Finetuning** adapts LLM parameters through continued training on domain-specific datasets, encoding EDA knowledge directly into model weights. This approach enables LLMs to internalize fundamental domain understanding, improving their comprehension of terminology, conventions, and domain-specific patterns. **Retrieval-augmented generation (RAG)** complements finetuning by maintaining external knowledge bases and dynamically retrieving relevant information at inference time. RAG systems can ground LLM responses in authoritative and up-to-date documentation sources,

This work is licensed under a Creative Commons Attribution International 4.0 License.

ISPD '26, March 15–18, 2026, Bonn, Germany
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2314-8/2026/03.
https://doi.org/10.1145/3764386.3779640

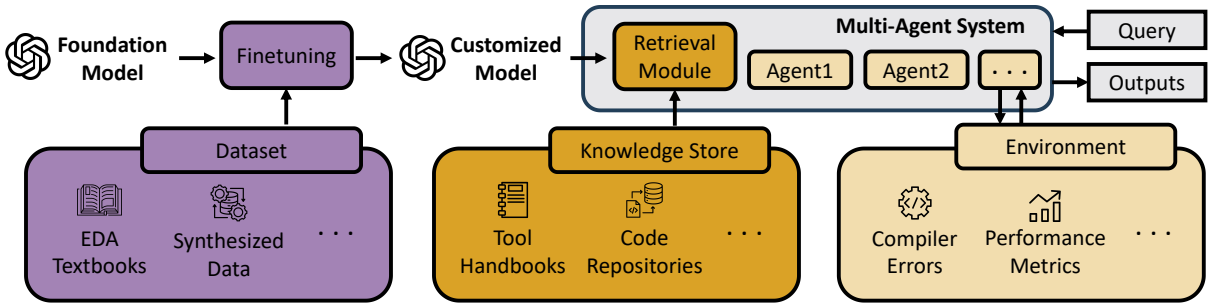


Figure 1: Domain knowledge injection for LLM-based EDA systems. Finetuning encodes knowledge from domain corpora into model parameters, while RAG retrieves from external knowledge bases at inference time. Both approaches integrate within agent frameworks for complex task handling.

reducing hallucinations and enabling verifiable outputs without retraining. **Multi-agent flow** addresses complex multi-step tasks by decomposing them into specialized subtasks and leveraging environment feedback for iterative refinement. By incorporating execution results from real-world environments, such as EDA tool outputs or human feedbacks, multi-agent systems can progressively improve solutions through feedback loops. These three techniques are complementary: finetuning provides foundational domain comprehension, RAG supplies up-to-date reference information, and multi-agent flow enables complex task handling with real-world feedback. Together, they form a comprehensive framework for injecting sufficient domain knowledge into LLM-based EDA systems.

This paper provides a focused survey of knowledge injection techniques for LLM-based EDA systems. We systematically examine finetuning approaches (Section 2), RAG systems (Section 3) and multi-agent flow (Section 4). To illustrate the potential of advanced RAG algorithms, we present a case study (Section 5) on graph-based RAG, demonstrating how graph structures can address challenges like cross-chunk information integration in EDA documentation QA. We conclude with insights and future directions (Section 6).

2 Finetuning

2.1 Methods

Finetuning is a fundamental technique to adapt pre-trained large language models (LLMs) to domain-specific tasks by continuing the training process on specialized datasets. While base LLMs are trained on general-purpose corpora, their knowledge of EDA is often limited and outdated. Finetuning addresses this limitation by injecting domain-specific knowledge directly into the model parameters, enabling the LLM to better understand EDA terminology, tool-specific syntax, and design patterns.

Formally, given a pre-trained model with parameters $\boldsymbol{\theta}_0$ and a domain-specific dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$ where x_i represents the input (e.g., task description or question) and y_i represents the target output (e.g., tool script or documentation answer), finetuning optimizes:

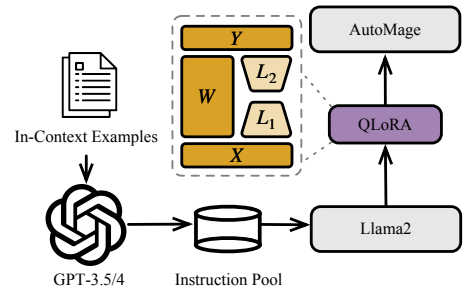
$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^{N} \mathcal{L}(f_{\boldsymbol{\theta}}(x_i), y_i) + \lambda \|\boldsymbol{\theta} - \boldsymbol{\theta}_0\|^2, \tag{1}$$

where $f_{\boldsymbol{\theta}}$ is the model with parameters $\boldsymbol{\theta}$, $\mathcal{L}$ is the task-specific loss function (typically cross-entropy for text generation), and the regularization term prevents catastrophic forgetting. In practice, parameter-efficient techniques such as low-rank adaptation (LoRA) [47] and QLoRA [48] are often employed to reduce computational costs by updating only a small subset of parameters.

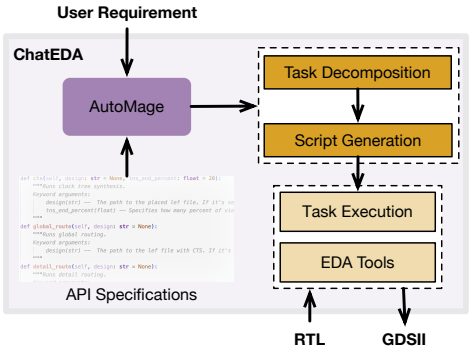
2.2 Representative Works

ChatEDA for EDA Script Generation. ChatEDA [26, 49] targets EDA script generation, automatically generating Python/Tcl scripts to automate physical design tools like OpenROAD, Innovus, and ICC. As shown in Figure 2, ChatEDA uses AutoMage as the controller to decompose tasks, generate scripts, and execute EDA tools. The RTL-to-GDSII physical design flow requires engineers to write intricate scripts that configure tool parameters, specify design constraints, and orchestrate multi-stage workflows including floorplanning, placement, clock tree synthesis, and routing. General-purpose LLMs struggle with this task because they lack knowledge of EDA-specific tool commands, frequently make syntax errors in tool-specific languages, and fail to satisfy domain constraints. To construct training data that addresses these gaps, ChatEDA employs a self-instruction paradigm powered by GPT models, as illustrated in Figure 2(b). Given seed examples of EDA scripts written by domain experts, GPT generates diverse task instructions describing script functionality, creating task-instruction-script triplets spanning the entire RTL-to-GDSII flow. Each triplet pairs a natural language task description (e.g., "Perform floorplanning with 70% core utilization and place macro blocks at chip corners") with the corresponding executable script in tool-specific syntax. The dataset incorporates task decomposition, breaking complex workflows into subtasks with modular script components, which teaches the model compositional reasoning over multi-step EDA processes. ChatEDA then applies instruction tuning on Llama2 with QLoRA to produce AutoMage, which significantly outperforms GPT-4 on EDA scripting benchmarks while using only 7B parameters, demonstrating that targeted domain finetuning is more effective than relying on model scale alone.

BetterV for RTL Generation. BetterV [5] targets RTL code generation, automatically generating Verilog modules from natural language descriptions or module definitions. Unlike general programming languages such as C and Python that have abundant training corpora, Verilog data is extremely scarce on the internet, which limits how well LLMs can learn hardware description semantics and syntax during pre-training. To address this data scarcity, BetterV employs cross-lingual knowledge transfer by leveraging the LLM's existing understanding of C programming. Training data is constructed in two forms: (1) Verilog-C translation pairs created using a V2C tool [50], enabling bidirectional translation training (Verilog↔C) that maps



(a) ChatEDA Framework



(b) AutoMage Instruction Tuning

Figure 2: ChatEDA [26, 49] system overview. ((a)) The framework uses AutoMage as controller to decompose tasks, generate scripts, and execute EDA tools. ((b)) Instruction tuning constructs domain-specific training data via GPT models and applies QLoRA for efficient finetuning.

Verilog semantics to equivalent C constructs the model already understands from pre-training; and (2) Verilog definition-body pairs for autocompletion training, where the model learns to complete module implementations given their interface definitions. This cross-lingual approach effectively uses C as a semantic bridge, allowing the model to transfer its rich understanding of programming concepts to the hardware description domain. The dataset is further augmented by using the finetuned LLM itself to generate synthetic Verilog modules with high temperature sampling, which are then filtered for syntactic correctness via EDA tools to ensure training quality. Additionally, BetterV introduces generative discriminators to enable controlled generation toward downstream task objectives. A smaller LLM is trained as a class-conditional language model using EDA tool feedback as labels (e.g., synthesis node counts, verification runtime), which then guides token-level generation via Bayes rule to steer toward implementations that are more friendly to downstream EDA processes. Experiments show BetterV outperforms GPT-4 on the VerilogEval benchmark while achieving approximately 47% reduction in synthesis netlist nodes and 22% reduction in SAT verification runtime compared to reference implementations.

3 Retrieval-Augmented Generation

3.1 Methods

Retrieval-augmented generation (RAG) addresses a fundamental limitation of large language models: their reliance solely on knowledge encoded in model parameters during pretraining, which can be outdated, incomplete, or insufficiently specialized [51]. RAG enhances LLMs by dynamically retrieving relevant information from external knowledge bases and incorporating it into the generation process. This approach is particularly valuable for LLM-based EDA systems, where tool documentation evolves rapidly with new releases and contains highly specialized information that general-purpose LLMs cannot adequately capture.

Conceptually, given a user query q, RAG first retrieves relevant documents from a knowledge base $\mathcal{K}$ based on semantic or lexical similarity. The retrieved documents are then augmented with the original query to form an enriched prompt, which is fed to the LLM for generation. This process can be formalized as:

$$a = \text{Generate}(\text{Augment}(q, \text{Retrieve}(q, \mathcal{K}))). \tag{2}$$

By grounding generation in retrieved context, RAG reduces hallucinations and enables LLMs to access knowledge beyond their training data. Additionally, RAG provides interpretable sources that engineers can verify, building trust in the system's outputs.

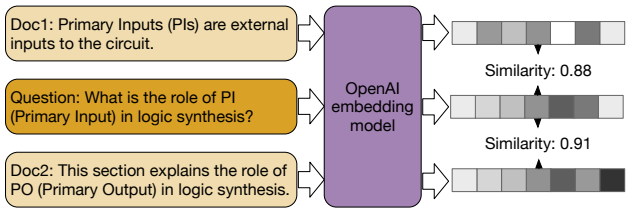
3.2 Representative Works

RAG-EDA and MM-GRADE for Document QA. RAG-EDA [22] and MM-GRADE [23] both leverage contrastive learning to enhance retrieval quality for EDA documentation question answering, addressing the fundamental challenge that standard embedding models fail to capture EDA-specific semantic relationships. Documentation QA requires retrieving precise information from extensive tool manuals, but general-purpose text embeddings demonstrate poor understanding of EDA terminologies, often confusing related but distinct concepts. As illustrated in Figure 3(a), a query about "Primary Input" may incorrectly retrieve documents about "Primary Output" due to similar sentence structures, even though their meanings differ significantly in EDA contexts. RAG-EDA addresses this through supervised contrastive learning that finetunes the embedding model to accurately distinguish EDA terms. The key principle is that sentences involving the same terminology should be positioned closely in the vector space regardless of structural differences, while sentences with similar structures but different terminologies should be separated. As shown in Figure 3(b), for each query x_i, training constructs a positive sample x_i^+ sharing the same EDA term and a hard negative sample x_i^- with similar structure but different terminology. The contrastive loss with in-batch sampling encourages proper semantic clustering:

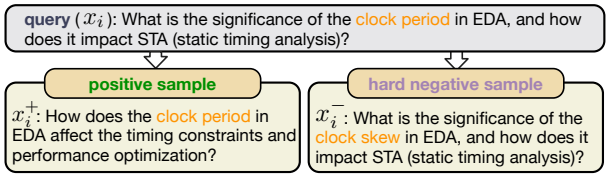
$$\mathcal{L}_{embed} = -\log \frac{e^{\text{sim}(x_i, x_i^+)/\tau}}{\sum_{j=1}^{M} \left(e^{\text{sim}(x_i, x_j^+)/\tau} + e^{\text{sim}(x_i, x_j^-)/\tau} \right)}, \tag{3}$$

where $\text{sim}(\cdot,\cdot)$ computes cosine similarity and τ is a temperature parameter. RAG-EDA similarly applies contrastive learning to train the reranker for filtering weakly-related documents, distilling GPT-4's fine filtering capabilities into a cross-encoder model.

MM-GRADE [23] extends this contrastive learning approach to handle multi-modal EDA documentation, where text-only queries often fail to fully describe complex issues and users include visual elements like GUI screenshots, design layouts, and error logs. MM-GRADE customizes the multi-modal embedding model through a bilevel hard negative mining (BHNM) strategy that addresses two complementary challenges. The first level applies document-level contrastive learning $\mathcal{L}_d^i$, which trains the retriever to position multi-modal queries close to relevant documents while separating them from irrelevant hard negatives. The second level addresses



(a) Fail Case of General Embedding



(b) Contrastive Learning Samples

Figure 3: RAG-EDA [22] domain-customized embedding. ((a)) General embedding models fail to distinguish EDA terms like Primary Input (PI) and Primary Output (PO), leading to incorrect retrieval. ((b)) Contrastive learning uses triplets of query, positive sample (same EDA term), and hard negative sample (similar structure, different term) to finetune embeddings.

a challenge unique to EDA tool documentation: the same screenshot image can trigger multiple distinct queries requiring different documents. For example, one screenshot may show both timing violations and placement errors, generating separate queries about `repair_timing` and `global_place` commands. Existing multi-modal retrievers encode these different queries at close proximity in the embedding space due to the shared image, degrading retrieval accuracy. To separate embeddings of different queries from the same image, MM-GRADE applies query-level contrastive learning $\mathcal{L}_q^i$, which treats other queries sharing the same image as hard negatives. The combined BHNM loss integrates both levels:

$$\mathcal{L} = \sum_{i=1}^{N} \mathcal{L}_d^i + \mathcal{L}_q^i. \tag{4}$$

HDLDebugger for HDL Debugging. HDLDebugger [14] employs RAG to provide contextual information for Hardware Description Language debugging, automatically correcting buggy HDL code given error messages from compilers. HDL debugging presents unique challenges: HDL codes are rarely made public due to privacy and commercial concerns, the syntax is complex with limited online resources, and debugging requires understanding both the error patterns and domain-specific correction strategies. Retrieving relevant information is critical for this task because LLMs lack sufficient HDL knowledge from pretraining, and similar bugs often have similar solutions that can guide the correction process. HDLDebugger constructs two complementary knowledge bases and retrieves from both:

- Document RAG retrieves from an error database containing detailed descriptions, underlying reasons, and suggested remedial strategies for each error code. Given an error message, the system parses it to extract constituent error codes and retrieves the corresponding diagnostic information, helping the LLM understand what went wrong and how to fix it.
- Code RAG retrieves similar buggy code instances along with their corrected versions from a code database. Using a BERT-LSTM model that encodes buggy code and error messages into semantic vectors combined with TF-IDF keyword vectors, the system identifies code with similar bug patterns. The retrieved buggy-correct pairs serve as in-context examples that demonstrate correction strategies for analogous errors.

The retrieved document information and code examples are combined with the input buggy code to form an enriched prompt for the LLM. HDLDebugger further applies retrieval-augmented finetuning that trains the LLM with both RAG contexts, achieving significant improvements over baselines including GPT-4 on industry-level HDL debugging benchmarks.

4 Multi-Agent Flow

4.1 Methods

While finetuning and RAG inject domain knowledge through static mechanisms, either encoding knowledge in model parameters or retrieving it from fixed databases, multi-agent flow approaches inject domain knowledge dynamically through interaction patterns and iterative refinement processes. This paradigm addresses EDA tasks requiring validation through synthesis tools, simulators, and domain-specific checkers, where design correctness cannot be determined from static information alone.

Multi-agent flow operates through two primary mechanisms. First, task decomposition breaks complex problems into specialized subtasks according to domain-specific workflow boundaries, with different processing stages handling distinct aspects of the problem. This approach reflects the insight that end-to-end models often struggle with complex EDA tasks that span multiple modalities or require diverse expertise, and that decomposing these tasks according to natural domain boundaries can significantly improve performance. Second, environment feedback injects knowledge through iterative interaction with EDA tools and simulators, enabling systems to learn from execution results and progressively refine solutions. Both mechanisms leverage the dynamic nature of EDA workflows, where intermediate results inform subsequent decisions and tool feedback validates design choices, to incorporate domain expertise into the generation process in ways that static approaches cannot achieve.

4.2 Representative Works

LayoutCopilot for Analog Layout. LayoutCopilot [36] exemplifies task decomposition by employing a multi-agent collaborative framework for interactive analog layout design, translating natural language instructions into executable layout tool commands. Analog layout design presents significant usability challenges: designers must master complex scripting commands and translate high-level design intents (e.g., "enhance the CMRR of this OTA") into precise tool operations, requiring deep knowledge of both circuit behavior and tool syntax. General-purpose LLMs struggle with this task due to prompt dilution—when a single agent handles knowledge retrieval, intent parsing, and code generation simultaneously, the extensive context from knowledge bases degrades performance on other subtasks. LayoutCopilot addresses this through task decomposition across six specialized agents, as illustrated in Figure 4. Given a designer's natural language request, a Classifier Agent first determines whether it is abstract (high-level intent) or concrete (direct tool operation). For abstract requests, an Analyzer Agent retrieves

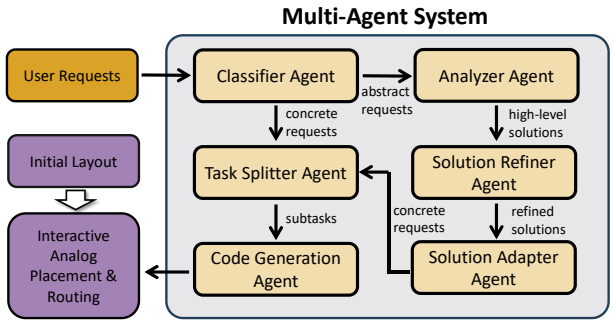


Figure 4: LayoutCopilot [36] multi-agent flow for interactive analog layout design. Six specialized agents collaborate to transform designer requests into executable commands: Classifier, Analyzer, Solution Refiner, and Solution Adapter agents handle abstract-to-concrete conversion, while Task Splitter and Code Generator agents produce tool commands.

domain knowledge from analog design books and tool manuals to generate high-level solutions; a Solution Refiner Agent then presents these options to designers and incorporates their feedback through iterative refinement; finally, a Solution Adapter Agent extracts specific parameters from circuit netlists to formulate concrete requests. The concrete requests are then processed by a Task Splitter Agent that decomposes them into subtasks corresponding to individual tool commands, and a Code Generator Agent that produces syntactically and logically correct command sequences for the layout tool. This multi-agent architecture enables each agent to focus on its specialized task without prompt dilution, while the human-in-the-loop design allows designers to guide the optimization process through natural language feedback. The framework achieves 93.75% overall accuracy in command generation and demonstrates effective layout optimization on real analog circuits.

AnalogCoder for Analog Circuit Design. AnalogCoder [32] exemplifies environment feedback by iteratively refining analog circuit designs through SPICE simulation results, automatically generating Python code that specifies circuit topology and device parameters. Analog circuit design presents unique challenges distinct from digital design: circuits must satisfy complex performance specifications across multiple metrics (gain, bandwidth, phase margin, power consumption), device parameters exhibit intricate interdependencies governed by analog design principles, and design correctness can only be verified through simulation rather than formal methods. General-purpose LLMs fail at this task because they lack understanding of analog design principles, cannot predict simulation outcomes, and have no mechanism to iteratively refine designs based on performance feedback. AnalogCoder addresses these limitations through a feedback-enhanced design flow that progressively refines circuit implementations. Given a design specification, the system generates initial circuit code and then improves it with four feedback types:

- Requirement check verifies that generated code satisfies basic structural constraints, ensuring the circuit contains all necessary components and connections specified in the design requirements.
- Simulation check analyzes operating point results from SPICE simulation to ensure MOSFETs operate in correct regions (saturation for amplifiers, triode for switches), catching fundamental biasing errors early.
- DC sweep check traces signal pathways by sweeping input voltages and observing output responses, verifying proper circuit connectivity and bias conditions across the operating range.
- Function check evaluates performance metrics (gain, bandwidth, phase margin) against target specifications, providing quantitative feedback on how close the design is to meeting requirements.

Each check provides specific diagnostic feedback that guides the LLM's next refinement iteration, enabling systematic debugging rather than random regeneration. Additionally, AnalogCoder maintains a circuit tool library that stores successfully designed subcircuits, enabling hierarchical composition where complex circuits can be built from verified building blocks. This feedback-driven approach transforms the LLM from a one-shot generator into an iterative problem solver that learns from simulation results within each design session, achieving significant improvements over baseline LLMs on analog design benchmarks.

5 Case Study: Graph-based RAG for EDA Documentation QA

The preceding sections have examined finetuning, RAG, and multi-agent flow as complementary approaches for injecting domain knowledge into LLM systems. While standard RAG techniques significantly improve EDA tool assistance capabilities, they primarily focus on retrieving information from individual document chunks. This case study demonstrates how graph-based techniques can enhance RAG systems to address more complex retrieval scenarios that require integrating information scattered across multiple chunks, a challenge that exposes fundamental limitations of traditional chunk-based retrieval.

5.1 Challenges and Motivation

Traditional chunk-based RAG systems excel at answering *local queries* where relevant information resides within a single document chunk. However, they struggle with *global queries* whose answers require synthesizing information distributed across multiple chunks. For example, understanding all methods to prevent cell modification during timing optimization may require combining information scattered across different sections or even separate documents: the handbook describes `dont_touch` command syntax, the user guide explains the timing optimization workflow, and release notes document version-specific behaviors. Standard retrieval mechanisms lack the ability to identify and integrate such scattered information, leading to incomplete or fragmented answers.

Knowledge graphs (KGs) offer a natural solution to this challenge. By representing documents as graphs with entities as nodes and relationships as edges, KG-based approaches enable finer-granularity retrieval that can traverse connections across document boundaries [52]. However, constructing high-quality KGs in specialized domains like EDA demands deep expertise; without it, automatically generated KGs suffer from two fundamental challenges, as illustrated in Figure 6. The first challenge is poor expression: structured KG triples cannot fully capture complex semantics. Simple subject-predicate-object triples cannot easily express conditional logic, procedural constraints, or context-dependent meanings common in EDA documentation. The second challenge is entity confusion: semantically equivalent entities may be misaligned across chunks. For example, `dont_touch` and `lock_settings` convey similar meanings but they are represented differently.

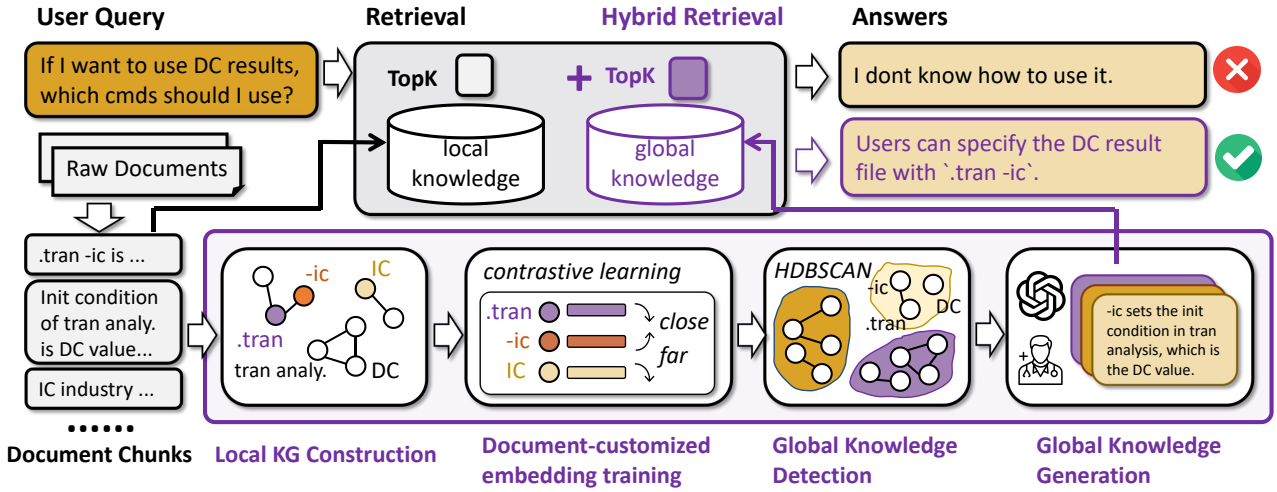


Figure 5: Architecture of graph-enhanced RAG. The system extracts global knowledge from local KGs and integrates it via hybrid retrieval. Red highlights indicate enhancement components.

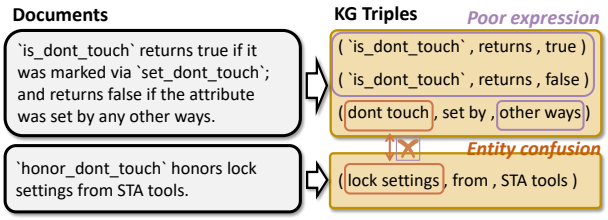


Figure 6: Challenges in automatic KG construction for EDA.

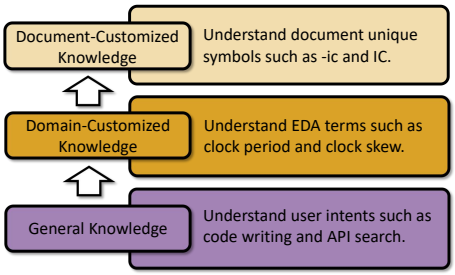


Figure 7: Hierarchy of knowledge customization in EDA.

To address entity confusion, we observe that the key limitation lies in the embedding model's lack of document-specific knowledge. Figure 7 formalizes a knowledge customization hierarchy. General knowledge enables basic task understanding. Domain-customized knowledge distinguishes EDA-specific concepts. However, document-customized knowledge is essential for resolving context-dependent ambiguities unique to specific documentation. For instance, it can distinguish `-ic` for initial conditions from `IC` referring to integrated circuits. This motivates our approach: train document-customized embeddings to resolve entity confusion, then leverage KGs to detect global knowledge while expressing it as text to avoid the poor expression problem.

5.2 Graph-Based RAG Method

Based on the above analysis, we propose a graph-enhanced RAG approach that leverages KGs to detect global knowledge while avoiding fully KG-based retrieval. Figure 5 illustrates the overall architecture, which comprises three core components: local KG construction extracts entities and relationships from individual chunks; document-customized embeddings are trained via contrastive learning to represent context-dependent entity concepts; and global knowledge detection identifies semantically related entities across chunks using clustering and produces textual summaries. During retrieval, a hybrid strategy retrieves both local chunks and global knowledge documents in parallel, enabling comprehensive answers from both perspectives. This plug-and-play design requires no modifications to the core RAG pipeline, as global knowledge documents are simply added alongside original chunks. Consequently, any existing chunk-based RAG system can benefit from cross-chunk reasoning capabilities without architectural changes.

Document-Customized Embeddings. Following the classic KG construction pipeline [53, 54], we split documents into chunks and use LLMs to extract entities and relationships from each chunk, producing local KGs $\mathcal{G} = \{g_d\}_{d \in \mathcal{D}}$ where g_d is the KG extracted from chunk d. The extraction identifies noun phrases as candidate entities and verb phrases as relationships, with the LLM filtering based on domain relevance. Although these KGs contain noise due to extraction errors and inconsistent naming, they encode approximate entity relationships that provide valuable training signals.

We employ supervised contrastive learning to train document-customized embeddings that distinguish semantically similar but contextually different entities. The key insight is that entities connected in the KG share contextual relationships, while entities that appear similar under general embeddings but lack KG connections are likely false positives. The training data collection proceeds as follows. For each entity e_i extracted from the local KGs, we construct positive samples from entities directly connected in any local KG:

$$\mathcal{P}(e_i) = \{e^+ \mid \exists g \in \mathcal{G},\ (e_i, e^+) \in g\}. \tag{5}$$

These positive pairs capture document-specific semantic relationships. For example, a command entity is paired with its option entities that appear in the same documentation chunk. We construct hard negatives through a three-step filtering process. First, we select candidates with high similarity under a general embedding model:

$$\tilde{\mathcal{N}}(e_i) = \{\tilde{e} \mid \text{sim}_{\text{general}}(e_i, \tilde{e}) > \delta\}, \tag{6}$$

identifying entities that would be confused without document-specific training. Second, we exclude entities within n hops in the KG, as these may have indirect but valid relationships:

$$\mathcal{N}'(e_i) = \{\tilde{e} \in \tilde{\mathcal{N}}(e_i) \mid d_{\mathcal{G}}(e_i, \tilde{e}) > n\}, \tag{7}$$

where $d_{\mathcal{G}}(\cdot,\cdot)$ denotes shortest-path distance. Third, we verify remaining candidates with an LLM to exclude contextually related entities, yielding final hard negatives $\mathcal{N}(e_i)$. Each training triplet (e_i, e_i^+, e_i^-) consists of an anchor entity, a positive sample from $\mathcal{P}(e_i)$, and a hard negative from $\mathcal{N}(e_i)$. The contrastive loss for a mini-batch of size M is:

$$\mathcal{L}_i = -\log \frac{e^{\text{sim}(\boldsymbol{v}_i, \boldsymbol{v}_i^+)/\tau}}{\sum_{j=1}^{M} \left(e^{\text{sim}(\boldsymbol{v}_i, \boldsymbol{v}_j^+)/\tau} + e^{\text{sim}(\boldsymbol{v}_i, \boldsymbol{v}_j^-)/\tau} \right)}, \tag{8}$$

where $\boldsymbol{v}_i, \boldsymbol{v}_i^+, \boldsymbol{v}_i^-$ are embedding vectors for the anchor, positive, and hard negative respectively, and τ is the temperature parameter.

Global Knowledge Detection and Generation. To address the poor expression problem, we extract global knowledge from KGs and represent it as textual summaries rather than structured graphs. The key insight is that while KG triples struggle to express complex semantics, they effectively capture entity co-occurrence patterns indicating topical relationships. By clustering entities that frequently appear together across chunks, we identify groups of related concepts that collectively encode global knowledge.

Using document-customized embeddings, we apply HDBSCAN clustering [55] to entity embeddings. HDBSCAN is particularly suitable because it automatically determines the number of clusters based on density, handles noise points that do not belong to any cluster, and does not require specifying cluster count a priori. The resulting clusters C_k represent semantically related entities spanning multiple chunks, with each cluster encoding a coherent topic within the documentation. For each cluster C_k, we collecet the corresponding document chunks $\mathcal{D}_k$ and prompt LLMs to synthesizes information from collected chunks, resolving contradictions and establishing clear relationships between clustered entities. This will generate a consolidated summary S_k capturing the global knowledge. Experts can optionally review these summaries to correct errors or inject missing information. Algorithm 1 details this process.

Hybrid Retrieval Integration. The generated global knowledge documents are integrated into existing RAG systems through a hybrid retrieval mechanism. During inference, two parallel retrieval streams are executed: the first performs traditional chunk retrieval over original documentation, capturing local knowledge with fine-grained detail; the second retrieves from global knowledge documents, providing cross-chunk context. Both streams use the document-customized embedding model, ensuring consistent semantic understanding.

Algorithm 1 Global Knowledge Detection and Generation

Input: Document chunks $\mathcal{D}$, local KGs $\mathcal{G}$, embedding model f, minimum cluster size m
Output: Global knowledge documents $\{S_k\}$

```
// Step 1: Entity extraction and embedding
Extract all entities E ← ⋃_{g∈G} entities(g)
Compute embeddings v_e ← f(e) for all e ∈ E
// Step 2: Density-based clustering
Apply HDBSCAN to {v_e} to obtain clusters {C_k}
Filter clusters: remove noise points and C_k where |C_k| < m
// Step 3: Global knowledge generation
for all cluster C_k do
    Collect source chunks D_k ← {d ∈ D | ∃e ∈ C_k, e ∈ g_d}
    Construct prompt with entity list C_k and chunk contents D_k
    S_k ← LLM(prompt)                    ▷ Generate summary
    S_k ← ExpertReview(S_k)              ▷ Optional refinement
end for
return {S_k}
```

5.3 Experimental Evaluation

Setup. For benchmarks, we evaluate on OpenROAD documentation [56], which consists of 291 chunks covering commands, tutorials, and usage guides for the open-source RTL-to-GDSII flow. Several open-source benchmarks exist for EDA documentation QA: EDA Corpus [29] provides general EDA QA pairs derived from GitHub issues and discussions, and ORD-QA [22] focuses on OpenROAD-specific queries with manually curated question-answer pairs. However, these existing benchmarks primarily contain single-chunk queries where answers can be found within individual document segments, limiting their ability to evaluate cross-chunk reasoning capabilities. To address this gap, we curate ORD-GQA, a benchmark of 120 global query pairs specifically designed to require information from multiple chunks. We generate candidate global queries by prompting LLMs with entity clusters and their source chunks, then manually verify and refine each query-answer pair to ensure quality. ORD-GQA exclusively consists of multi-chunk queries targeting global knowledge.

For baselines, we compare against EDA documentation QA systems including ORAssistant [25] and RAG-EDA [22], which represent state-of-the-art chunk-based RAG approaches with domain-specific customization. We also evaluate general-purpose graph-based RAG methods including GraphRAG [53], LightRAG [54], and ToG [57]. We exclude OpenROAD-Assistant [28] from comparison because it was originally designed to retrieve sample QA pairs rather than raw document chunks, and its generator was fine-tuned accordingly; directly applying this fine-tuned generator to chunk retrieval introduces a format mismatch, leading to degraded performance.

For metrics, we employ three complementary measures. BLEU [58] measures n-gram precision between generated and reference answers, capturing lexical accuracy. ROUGE-L [59] computes the longest common subsequence ratio, evaluating structural similarity and recall. UniEval [60] uses a pretrained evaluator to assess semantic quality including coherence, consistency, and relevance, providing a more comprehensive measure than lexical metrics alone.

From the OpenROAD documentation, our approach extracted 3,507 entities and 5,670 relationships, producing 7,907 training triplets. HDBSCAN identified 167 clusters, generating 112 global knowledge documents. Figure 8 illustrates an example of how two document chunks are decomposed into KG elements and then reorganized into global knowledge through entity clustering and LLM summarization.

RAG system	EDA Corpus			ORD-QA			ORD-GQA		
	BLEU	ROUGE-L	UniEval	BLEU	ROUGE-L	UniEval	BLEU	ROUGE-L	UniEval
GraphRAG	0.008	0.098	0.619	0.020	0.156	0.615	0.025	0.156	0.612
LightRAG	0.012	0.102	0.631	0.029	0.164	0.713	0.062	0.203	0.798
ToG	0.014	0.137	0.621	0.018	0.222	0.690	0.061	0.212	0.703
ORAssistant	**0.010**	0.123	0.686	0.044	0.189	0.703	0.034	0.190	0.786
ORAssistant + Ours	0.009	**0.128**	**0.690**	**0.048**	**0.192**	**0.732**	**0.047**	**0.206**	**0.812**
RAG-EDA	0.035	**0.223**	0.609	0.098	0.302	0.729	0.040	0.228	0.756
RAG-EDA + Ours	**0.036**	0.218	**0.609**	**0.104**	**0.324**	**0.768**	**0.108**	**0.312**	**0.800**

Table 1: Answer quality comparison. Graph-based baselines (GraphRAG, LightRAG, ToG) show limited performance. Our approach (+Ours) improves global query performance (ORD-GQA) while maintaining local query quality.

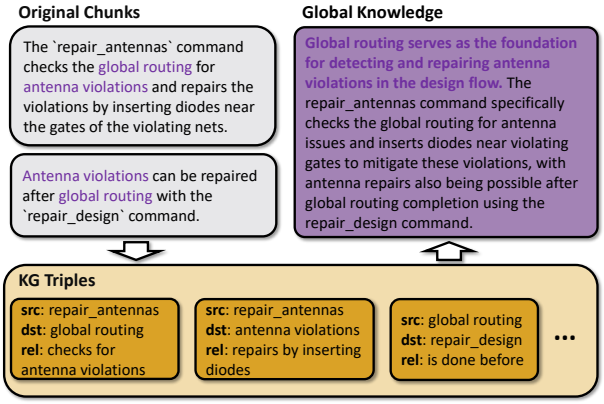


Figure 8: Example of global knowledge generation: two chunks are decomposed into KG elements and reorganized into a unified summary.

Answer Quality. Table 1 presents results across all benchmarks. A key observation is that without our enhancement, ORD-GQA performance is consistently lowest across all systems, confirming that standard RAG architectures struggle with global queries requiring cross-chunk reasoning. After integrating our approach, ORD-GQA performance improves significantly. Importantly, performance on existing benchmarks remains stable or slightly improves, demonstrating that our approach captures global knowledge without sacrificing local detail.

Table 2 provides a fine-grained analysis by stratifying performance according to the number of ground-truth chunks required to answer each query. For single-chunk queries where local knowledge suffices, both local and hybrid retrieval perform similarly. However, as the required chunk count increases, hybrid retrieval demonstrates increasingly clear advantages. For queries requiring 4 or more chunks, hybrid retrieval achieves much better performance compared to that of local retrieval, validating effectiveness for global queries.

Graph-based baselines exhibit limited performance compared to chunk-based systems enhanced with our approach. GraphRAG constructs communities via graph density metrics, but this purely structural approach lacks semantic information, especially domain-customized knowledge; consequently, its communities cannot effectively gather the related information required for global queries in specialized EDA contexts. LightRAG uses lightweight graph indexing but lacks effective semantic reranking methods, leading to high noise in retrieval results. ToG employs iterative path-based reasoning over the KG, but its reasoning can be misled without accurate KG schemas.

Retrieval	Metric	Chunks		
		# 1	# 2-3	# 4+
Local	BLEU	**0.052**	0.032	0.023
	ROUGE-L	0.240	0.189	0.175
	UniEval	0.691	0.723	0.771
Hybrid	BLEU	0.050	**0.041**	**0.032**
	ROUGE-L	**0.246**	**0.198**	**0.183**
	UniEval	**0.712**	**0.748**	**0.794**

Table 2: Performance stratified by ground-truth chunk count. Hybrid retrieval shows increasing advantage as queries require more chunks.

These challenges are common in specialized domains where automatic KG construction is inherently noisy. Our approach mitigates these issues by using KGs primarily for knowledge detection rather than direct retrieval, and expressing global knowledge as text to preserve expressiveness.

6 Conclusion

This paper surveyed knowledge injection techniques for LLM-based EDA systems, examining how domain-specific knowledge can be systematically infused into LLM systems to enable effective documentation QA and script generation. We analyzed finetuning approaches that encode EDA knowledge into model parameters through training on domain corpora and synthetic data, enabling intrinsic understanding of tool-specific terminology and conventions. We examined RAG systems that dynamically retrieve from external knowledge bases, highlighting that domain customization is essential for handling EDA-specific semantics. We discussed multi-agent flow approaches that decompose complex tasks across specialized agents and leverage environment feedback for iterative refinement, enabling dynamic knowledge injection through interaction patterns that static approaches cannot achieve. Our case study on graph-based RAG demonstrated how knowledge graph structures can capture global knowledge spanning multiple document chunks, addressing cross-chunk reasoning challenges in EDA documentation. Advanced techniques like graph-based RAG are particularly promising for EDA tasks because the graph construction process inherently encodes domain knowledge through entity extraction and relationship modeling. Together, these three complementary techniques provide pathways for building effective LLM systems for EDA: finetuning for permanent domain adaptation, customized RAG for dynamic and interpretable knowledge access, and multi-agent flow for complex task handling with real-world feedback.

References

[1] C. Zhong *et al.*, "LLM4EDA: Emerging Progress in Large Language Models for Electronic Design Automation," *arXiv preprint*, 2024.

[2] F. Liu *et al.*, "LLM4Logic: Large Language Models for Logic Synthesis," *arXiv preprint*, 2024.

[3] Z. He, Y. Pu, H. Wu, Y. Qiu, T. Qiu, and B. Yu, "Large Language Models for EDA: From Assistants to Agents," *Foundations and Trends in Electronic Design Automation*, 2025.

[4] 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," *ACM TODAES*, 2023.

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

[6] M. Liu, N. Pinckney, B. Khailany, and H. Ren, "VerilogEval: Evaluating Large Language Models for Verilog Code Generation," in *Proc. ICCAD*, 2023.

[7] 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," in *Proc. DATE*, 2023.

[8] Y. Lu, S. Liu, Q. Zhang, and Z. Xie, "RTLLM: An Open-Source Benchmark for Design RTL Generation with Large Language Models," in *Proc. ASPDAC*, 2024.

[9] J. Blocklove, S. Garg, R. Karri, and H. Pearce, "Chip-Chat: Challenges and Opportunities in Conversational Hardware Design," in *Proc. MLCAD*, 2023.

[10] 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," *arXiv preprint*, 2023.

[11] X. Yao, W. Zhao, Q. Sun, C. Zhuo, and B. Yu, "High-Level Synthesis Directives Design Optimization via Large Language Model," *arXiv preprint*, 2024.

[12] 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," in *Proc. ICCAD*, 2023.

[13] X. Yao, Y. Wang, X. Li, Y. Lian, R. Chen, L. Chen, M. Yuan, H. Xu, and B. Yu, "RTLRewriter: Methodologies for Large Models aided RTL Code Optimization," in *Proc. ICCAD*, 2024.

[14] 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," *ACM TODAES*, 2024.

[15] Y. Tsai, M. Liu, and H. Ren, "RTLFixer: Automatically Fixing RTL Syntax Errors with Large Language Models," in *Proc. DAC*, 2024.

[16] S. Thakur, J. Blocklove, H. Pearce, B. Tan, S. Garg, and R. Karri, "AutoChip: Automating HDL Generation using LLM Feedback," *arXiv preprint*, 2023.

[17] Z. Yan, W. Fang, M. Li, M. Li, S. Liu, Z. Xie, and H. Zhang, "AssertLLM: Generating Hardware Verification Assertions from Design Specifications via Multi-LLMs," in *Proc. ASPDAC*, 2025.

[18] R. Kande, H. Pearce, B. Tan, B. Dolan-Gavitt, S. Thakur, R. Karri, and J. Rajendran, "LLM-Assisted Generation of Hardware Assertions," *arXiv preprint*, 2023.

[19] S. Paria, A. Dasgupta, and S. Bhunia, "DIVAS: An LLM-Based End-to-End Framework for SoC Security Analysis and Policy-Based Protection," *arXiv preprint*, 2023.

[20] J. Chaudhuri, D. Thapar, A. Chaudhuri, F. Firouzi, and K. Chakrabarty, "SPICED: Syntactical Bug and Trojan Pattern Identification in A/MS Circuits using LLM-Enhanced Detection," *arXiv preprint*, 2024.

[21] B. Ahmad, S. Thakur, B. Tan, R. Karri, and H. Pearce, "Fixing Hardware Security Bugs with Large Language Models," *arXiv preprint*, 2023.

[22] Y. Pu, Z. He, T. Qiu, H. Wu, and B. Yu, "Customized Retrieval Augmented Generation and Benchmarking for EDA Tool Documentation QA," in *Proc. ICCAD*, 2024.

[23] Y. Pu, Z. He, S. Lin, J. Qin, X. Zhang, H. Han, H. Zheng, Y. Jiang, C. Zhuo, Q. Sun, D. Pan, and B. Yu, "MM-GRADE: A Multi-Modal EDA Tool Documentation QA Framework Leveraging Retrieval Augmented Generation," in *Proc. MLCAD*, 2024.

[24] L. Shi, M. Kazda, B. Sears, N. Shropshire, and R. Puri, "Ask-EDA: A Design Assistant Empowered by LLM, Hybrid RAG and Abbreviation De-hallucination," in *IEEE International Workshop on LLM-Aided Design*, 2024.

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

[26] 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," *IEEE TCAD*, 2024.

[27] 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," *arXiv preprint*, 2023.

[28] 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.

[29] B.-Y. Wu, U. Sharma, S. R. D. Kankipati, A. Yadav, B. K. George, S. R. Guntupalli, A. Rovinski, and V. A. Chhabria, "EDA Corpus: A Large Language Model Dataset for Enhanced Interaction with OpenROAD," in *IEEE International Workshop on LLM-Aided Design*, 2024.

[30] M. Abdelatty and S. Reda, "HDLCopilot: Hardware Design Library Querying with Natural Language," *arXiv preprint*, 2024.

[31] X. Wang, B. Han, Z. Tai, J. Tian, Y. Wang, J. Yan, and Y. Tian, "New Interaction Paradigm for Complex EDA Software Leveraging GPT," in *Proc. ICML*, 2025.

[32] Y. Lai, S. Lee, G. Chen, S. Poddar, M. Hu, D. Z. Pan, and P. Luo, "AnalogCoder: Analog Circuit Design via Training-Free Code Generation," in *Proc. AAAI*, 2024.

[33] Y. Yin, Y. Wang, B. Xu, and P. Li, "ADO-LLM: Analog Design Bayesian Optimization with In-Context Learning of Large Language Models," in *Proc. ICCAD*, 2024.

[34] C. Liu, W. Chen, A. Peng, Y. Du, L. Du, and J. Yang, "AmpAgent: An LLM-based Multi-Agent System for Multi-stage Amplifier Schematic Design from Literature for Process and Performance Porting," *arXiv preprint*, 2024.

[35] A. Hammoud, C. Goyal, S. Pathen, A. Dai, A. Li, G. Kielian, and M. Saligane, "Human Language to Analog Layout Using GLayout Layout Automation Framework," in *Proc. MLCAD*, 2024.

[36] B. Wu *et al.*, "LayoutCopilot: An LLM-powered Multi-agent Collaborative Framework for Interactive Analog Layout Design," *IEEE TCAD*, 2024.

[37] Y. Jiang, X. Lu, Q. Jin, Q. Sun, H. Wu, and C. Zhuo, "FabGPT: An Efficient Large Multimodal Model for Complex Wafer Defect Knowledge Queries," in *Proc. ICCAD*, 2024.

[38] M. Dupree, M. J. Yang, Y. Zeng, and L.-C. Wang, "IEA-Plot: Conducting Wafer-Based Data Analytics Through Chat," in *Proc. ITC*, 2024.

[39] T. Li, Z. Tang, T. Wu, B. Yu, J. Yu, and H. Geng, "LLM-SRAF: Sub-Resolution Assist Feature Generation Using Large Language Model," in *Proc. DATE*, 2025.

[40] G. Chen, H. Yang, B. Yu, and H. Ren, "Intelligent OPC Engineer Assistant for Semiconductor Manufacturing," *arXiv preprint*, 2024.

[41] Z. Wang, Y. Shen, X. Yao, W. Zhao, Y. Bai, F. Farnia, and B. Yu, "ChatPattern: Layout Pattern Customization via Natural Language," in *Proc. DAC*, 2024.

[42] M. Li, W. Fang, Q. Zhang, and Z. Xie, "SpecLLM: Exploring Generation and Review of VLSI Design Specification with Large Language Model," in *Proc. ISEDA*, 2024.

[43] S. Qiu, B. Tan, and H. Pearce, "LLM-aided Explanations of EDA Synthesis Errors," in *IEEE International Workshop on LLM-Aided Design*, 2024.

[44] N. Xu, Z. Zhang, L. Qi, W. Wang, C. Zhang, Z. Ren, H. Zhang, X. Cheng, Y. Zhang, Z. Liu *et al.*, "ChipExpert: The Open-Source Integrated-Circuit-Design-Specific Large Language Model," *arXiv preprint*, 2024.

[45] D. Luo, T. Li, X. Li, Q. Sun, C. Zhuo, B. Yu, J. Yu, and H. Geng, "LLM-Augmented Multi-Modal Fusion for SoC Design Space Exploration," in *Proc. ICCAD*, 2025.

[46] Z. Jiang, Q. Zhang, C. Liu, L. Cheng, H. Li, and X. Li, "IICPilot: An Intelligent Integrated Circuit Backend Design Framework Using Open EDA," *arXiv preprint*, 2024.

[47] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, "LoRA: Low-Rank Adaptation of Large Language Models," in *Proc. ICLR*, 2022.

[48] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLoRA: Efficient Finetuning of Quantized LLMs," in *Proc. NeurIPS*, 2023.

[49] 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," in *Proc. MLCAD*, 2023.

[50] R. Mukherjee, M. Tautschnig, and D. Kroening, "v2c – A Verilog to C Translator," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, 2016.

[51] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, M. Wang, and H. Wang, "Retrieval-Augmented Generation for Large Language Models: A Survey," *arXiv preprint*, 2024.

[52] B. Peng, Y. Zhu, Y. Liu, X. Bo, H. Shi, C. Hong, Y. Zhang, and S. Tang, "Graph Retrieval-Augmented Generation: A Survey," *Information Systems*, 2025.

[53] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, and J. Larson, "From Local to Global: A Graph RAG Approach to Query-Focused Summarization," *arXiv preprint*, 2024.

[54] Z. Guo, L. Xia, Y. Yu, T. Ao, and C. Huang, "LightRAG: Simple and Fast Retrieval-Augmented Generation," in *Proc. EMNLP*, 2025.

[55] M. B. Claudia Malzer, "A Hybrid Approach To Hierarchical Density-based Cluster Selection," *IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, 2020.

[56] T. Ajayi, V. A. Chhabria, M. Fogaça, S. Hashemi, A. Hosny, A. B. Kahng, M. Kim, J. Lee, U. Mallappa, M. Neseem, G. Pradipta, S. Reda, M. Saligane, S. S. Sapatnekar, C. Sechen, M. Shalan, W. Swartz, L. Wang, Z. Wang, M. Woo, and B. Xu, "Toward an Open-Source Digital Flow: First Learnings from the OpenROAD Project," in *Proc. DAC*, 2019.

[57] J. Sun, C. Xu, L. Tang, S. Wang, C. Lin, Y. Gong, L. M. Ni, H.-Y. Shum, and J. Guo, "Think-on-Graph: Deep and Responsible Reasoning of Large Language Model on Knowledge Graph," in *Proc. ICLR*, 2024.

[58] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "BLEU: A Method for Automatic Evaluation of Machine Translation," in *Proc. ACL*, 2002.

[59] C.-Y. Lin, "ROUGE: A Package for Automatic Evaluation of Summaries," in *Proc. ACL*, 2004.

[60] 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," in *Proc. EMNLP*, 2022.