DESIGNCON® 2026
WHERE THE CHIP MEETS THE BOARD

Agentic AI for SI/PI: From Concept to Design Execution

Priyank Kashyap, Hewlett Packard Enterprise
(priyank.kashyap@hpe.com)

Nirjhor Rouf, North Carolina State University
(nrouf2@ncsu.edu)

Yuejiang Wen, Hewlett Packard Enterprise
(jackson.wen@hpe.com)

Yongjin Choi, Hewlett Packard Enterprise
(yongjin.choi@hpe.com)

Paul Franzon, North Carolina State University
(paulf@ncsu.edu)

Chris Cheng, Hewlett Packard Enterprise
(chris.cheng@hpe.com)

Abstract

To complete a high-speed design, engineers need to understand a significant amount of design data, with 1000+ page documents not uncommon. In addition to the high design complexity, there is an ever-increasing design speed, which requires simultaneously optimizing design parameters, resulting in high-dimensional problems. Finally, introducing routing constraints compounds the design complexity, as there can be tens of thousands of constraints in a complex PCB design. All of the aforementioned problems clearly highlight the need for an end-to-end flow with a simplified interface.

This paper outlines a multi-agentic AI approach to high-speed SI/PI design, from concept to final execution, in which multiple AI agents perform dedicated design tasks to meet this need. A retrieval agent ingests domain knowledge from design documents, an optimization agent searches for optimal design solutions within the target range, and a constraint agent interacts with the SI/PI tool domain to generate routeable constraints.

The retrieval agent enables a fully updateable, flexible way to incorporate design documents into the design flow. At the same time, the optimization agent leverages human-in-the-loop to ensure successful execution. The constraint agent will turn the optimized results into design constraints for the PCB router and finish the design. Such an AI-driven agentic flow serves as a co-pilot to assist engineers with these design tasks. Our key innovation centers on enabling a multi-agent flow to perform optimization rather than merely executing mundane simulation and reporting tasks.

Author(s) Biography

Priyank Kashyap is a Senior AI/ML Engineer in the Storage Division of Hewlett-Packard Enterprise, where he works at the intersection of AI/ML and SI/PI. He received a Ph.D. and Master's in Computer Engineering from North Carolina State University in 2023 and 2022, respectively. He has previously worked at Cadence, Samsung Semiconductors, and Synopsys in applying AI/ML to numerous domains. His current research interests lie in applying machine learning to electronic system design and hardware security.

Nirjhor Rouf earned his B.Sc. degree from Brac University, Bangladesh, and completed his M.Sc. in Electrical Engineering from the University of Texas at Arlington. Currently, he is pursuing his Ph.D. degree under Prof. Paul Franzon's supervision at North Carolina State University. His research focuses on applications of Large Language Models (LLM) in the field of Electronic Design Automation (EDA).

Yuejiang Wen received the B.E. degree in optoelectronic information science and engineering and the M.Sc. degree in optical engineering from the University Electronic Science and Technology of China (UESTC), Chengdu, China, in 2012 and 2015, respectively, and the second M.Sc. degree in electrical engineering from Clemson University, Clemson, SC, USA, in 2018. He is currently pursuing the Ph.D. degree in electrical engineering with North Carolina State University, Raleigh, USA.

Yongjin Choi is a Master Technologist at the Storage Division of Hewlett-Packard Enterprise, where he leads the system failure prediction and SerDes modeling based on machine learning approach. He received a Ph.D. degree in electrical engineering from North Carolina State University in 2010. His technical interest includes applying machine-learning methodology to electronic system design.

Paul D. Franzon received the B.S., B.E.(hons) and Ph.D. degrees from The University of Adelaide, Adelaide, SA, Australia, in 1983, 1984 and 1989. He has been with AT\&T Bell Laboratories, Holmdel, NJ, USA; DSTO Australia, Salisbury, SA, Australia; and Australia Telecom, Adelaide. He co-founded four companies, Communica, Adelaide; LightSpin Technologies, Bethesda, MD, USA; and Indago, Raleigh, NC, and Polymer Braille Inc, Raleigh, NC, USA. He served with the Australian Army Reserve for 13 years as an Infantry Soldier and Officer. He is currently the Cirrus Logic Distinguished Professor, and Director of Graduate Programs in electrical and computer engineering with North Carolina State University, Raleigh. He has led several major efforts and authored over 300 papers in these areas. His current research interests include the technology, design and EDA of complex microsystems incorporating VLSI, advanced packaging, machine learning, and nanoelectronics. He received the NSF Young Investigators Award in 1993. He was selected to join the NCSU Academy of Outstanding Teachers in 2001, and received the Alcoa Research Award in 2005.

Chris Cheng is a Distinguished Technologist at the Storage Division of Hewlett-Packard Enterprise. He is responsible for managing all high speed and electrical designs within the Storage Division. He also held senior engineering positions in Sun Microsystems where he developed the original GTL system bus with Bill Gunning. He was a Principal

Engineer in Intel where he led high speed processor bus design team. He was the first hardware engineer in 3PAR and guided their high speed design effort until it was acquired by Hewlett Packard.

Introduction

Since the public release of ChatGPT in 2022, the number of large language models (LLMs) and their performance on benchmark and real-world applications have grown tremendously. For instance, LLM context windows of 10^4 to 10^6 are now commonplace for both open and closed models. The larger context window, on its own, enables LLMs to ingest and process large volumes of data for downstream tasks. Further, the multimodal capabilities of LLMs have improved, enabling them to work with various data types, such as text and images [1]. At the same time, the agentic capabilities of LLMs, which allow models to reason with provided tools (such as APIs), act by executing APIs, observe the results of execution, and then refine the flow for subsequent execution, have enabled LLMs to automate workflows across numerous domains, including EDA.

There has been tremendous work to integrate LLMs into EDA flows. We have seen applications of LLMs where researchers used them conversationally to co-design an 8-bit microprocessor [2]. Other approaches rely on finetuning LLMs; for instance, ChipNeMo is a chatbot assistant focused on engineering productivity that generates EDA scripts and performs bug summarization and analysis [3]. Similarly, ChatEDA demonstrates an RTL-to-GDSII flow orchestrated by finetuning an LLM to drive an EDA tool using an API [4]. On the contrary, low-compute approaches relying on retrieval-augmented generation (RAG) have also found broad adoption [5, 6]. Ask-EDA proposes a hybrid RAG technique to create a domain-enhanced LLM, which aims to tackle hallucinations, a fundamental issue with LLMs [5]. RAG frameworks were also helpful in generating RTL code without the overhead of training LLMs [6].

Recently, there has been a push for agentic approaches that integrate tools directly with LLMs to address LLMs' limitations in reasoning. Agentic workflows enable LLMs to execute tools to perform different tasks, observe outcomes, and iterate. Within EDA, Gadde et al. use a hybrid approach that combines an agentic with a human-in-the-loop approach to perform end-to-end hardware design and verification [7]. Firouzi et al. present an LLM-driven framework consisting of specialized agents for chip design [8]. Lastly, Ghose et al. show promising results using the agentic LLM approach for design optimization. The proposed method uses an open-source chip design flow that outperforms traditional Bayesian Optimization (BO) with fewer flow runs [9].

However, most of the work focuses on the chip side, making it hard to evaluate whether these models can be applicable to the PCB/system side, which has numerous design stages. These stages range from extracting constraints from platform design guides (PDGs) and setting up routing tools to optimizing and analyzing structures that contribute to high loss. With complex workflows in mind, we explore a multi-agent framework to support high-performance system design.

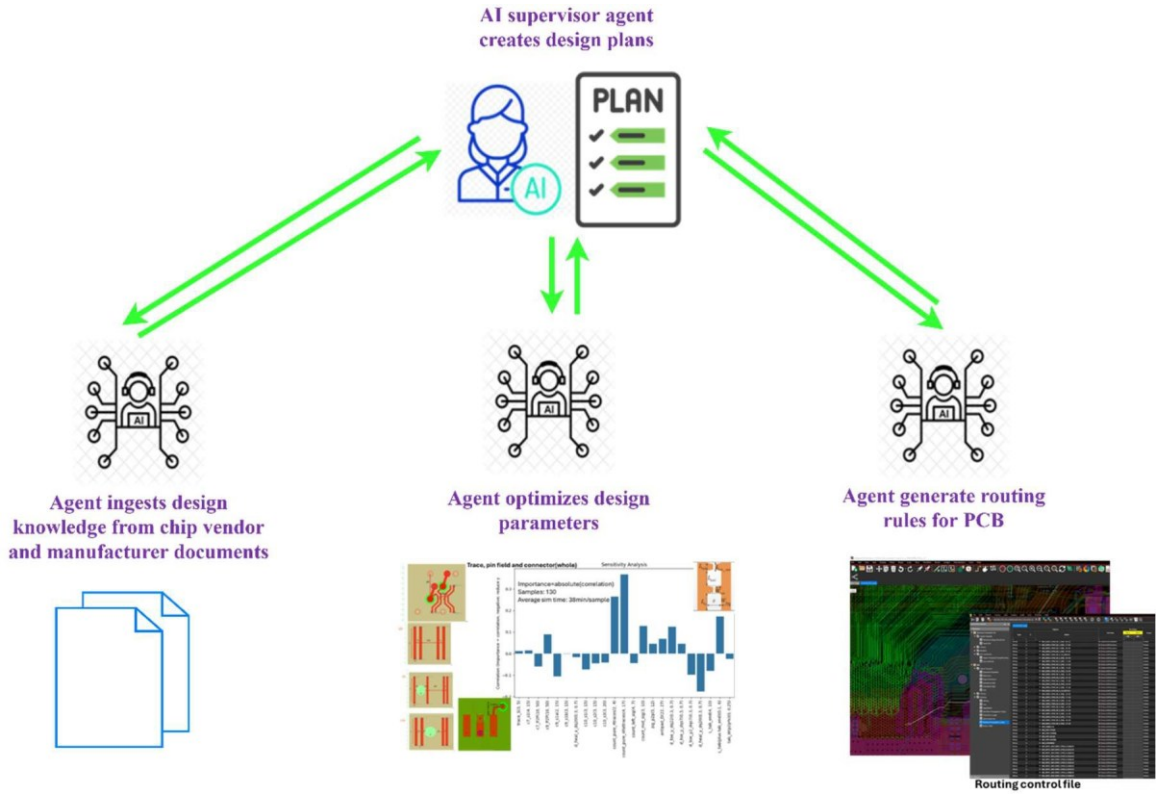

Figure 1. Overview of the proposed multi-agent framework. The RAG agent retrieves information, which serves as the starting point for either constraint for routing or for optim-tool agent which run EDA tools and analysis.

Figure 1 shows the proposed multi-agent flow with three dedicated agents. At the core of our work is our retrieval agent, which gathers information from PDGs to inform the next steps. The following steps use either the constraint agent, which sets up routing constraints (physical/electrical/spacing), or the optim-tool agent, which can optimize and analyze problems such as EM structures or SerDes parameters. With that in mind, the contributions of this work are as follows:

- We present a flexible multi-agentic flow that can serve as the basis for many EM and SI tasks.
- We present a multimodal retrieval agent that ingests large volumes of design information and extracts the necessary information to initiate the design process.
- The design information from the retrieval agent then flows either to the constraint agent to set routing parameters or our optim-tool agent.
- We present optim-tool, a flexible optimization agent that leverages human-designed algorithms to optimize performance from routing topology to final SerDes performance tuning.

The rest of the paper follows this structure. Section 2 provides the background on how LLMs work and the techniques that enable our proposed approach. Section 3 details our proposed multi-agent framework and focuses on the implementation and capabilities of each agent. Section 4 discusses libraries and other considerations for designing our multi-agent system. Section 5 presents our experiments, which demonstrate the LLM's ability to plan and execute a complex task, such as generating routing constraints. It dives deeper

into each agent's performance, explores the trade-off between direct and agentic optimization, and presents optimization results for a pin-field and a SerDes. Lastly, Section 6 concludes the paper and presents directions for future work.

Background

This section provides readers with the background they need to understand the proposed flow for various tasks.

LLMs, at their core, use transformer models, which enable them to attend to different parts of a sentence/query. However, different models use different transformer variants to extract better performance from their LLMs [10]. Given that most LLMs are transformer-based, they typically have two stages of training: pre-training and post-training. In the subsequent paragraphs, we explore these training regimes; however, the definitions are rudimentary, and there is much more complexity beneath the surface.

During the pre-training phase, LLMs aim to predict the next token autoregressively, one at a time, using prior predictions to inform subsequent word predictions based on the training corpus. Unlike words, tokens aim to be an efficient representation of words and punctuation by combining frequently occurring terms. Therefore, a sentence that contains 10 words may contain a lower number of tokens, depending on the frequency of words in the training corpus. At this point, the trained LLM predicts the next token, which is not helpful for general tasks and is generally called a base model.

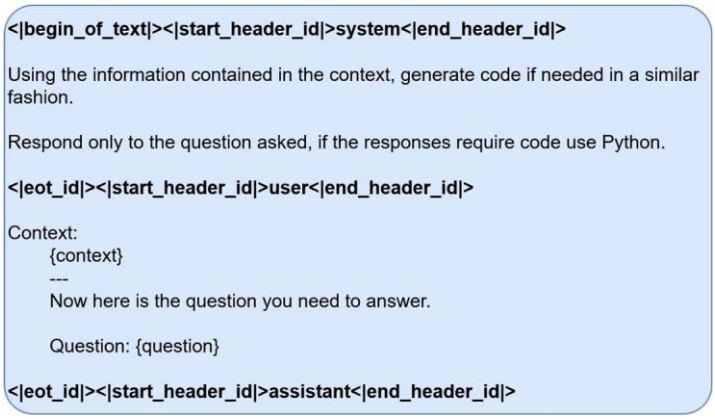

Figure 2. Showing the system prompt for the retrieval agent and where the additional context is provided.

Thus, during the post-training phase, the base model uses supervised fine-tuning (SFT) and reinforcement learning with human feedback (RLHF) to become a specialized model. During SFT, the training provides the LLM with input-output examples that inform what to say and how to format content. It is during this stage that specialized tags (|start_header_id|, |eot_id|, etc.), such as the one shown in Figure 2, are set, enabling the LLM to act as a chatbot with dedicated responses. Unlike SFT, RLHF teaches the model how to behave by using human reviewers to determine and learn subjective preferences. Post-training can be one of SFT, RLHF, or a combination of both [11].

Since the release of ChatGPT, researchers have devised numerous tricks to improve LLM performance. One such idea is chain-of-thought (CoT), which encourages the LLM to generate intermediate reasoning steps rather than jumping to a final answer, thereby improving LLMs' ability to solve various tasks [12]. Building upon this is the reasoning and action framework (ReACT), which encourages the LLM to create a plan that splits a complex task into smaller tasks that rely on external tools [13]. Based on the tool's response, the LLM refines its plan and iterates until it successfully solves the task or reaches a user-defined stop condition.

Proposed Methodology

This section discusses the proposed approach. It first presents the overall multi-agent flow and the interactions among agents. It then dives deeper into the role that each agent plays. Lastly, the section presents example flows that lead from a PDG to either a constraint or an optimization task.

Multi-Agent Flow

The goal of our multi-agentic approach is to have a framework that spans from design requirements to optimization and, finally, to routing constraints. To do so, our flow consists of 3 worker agents and one supervisor agent to manage them. The three agents are a RAG-based retrieval agent, a constraint-generation agent for routing constraints, and an optimization agent (optim-tool). We lightly couple the agents so that data can flow from the supervisor to the sub-agents, but not directly between them. Figure 3 shows the complete set of agents and the supervisor's flow.

In our flow, the retrieval agent provides domain-specific knowledge and effectively serves as the starting point for our other flows. The constraint-generation agent uses the provided information to generate routing constraints. Lastly, the optim-tool agent is flexible and capable of optimizing and analyzing a wide range of tasks. We cover each agent more in-depth in the following subsections.

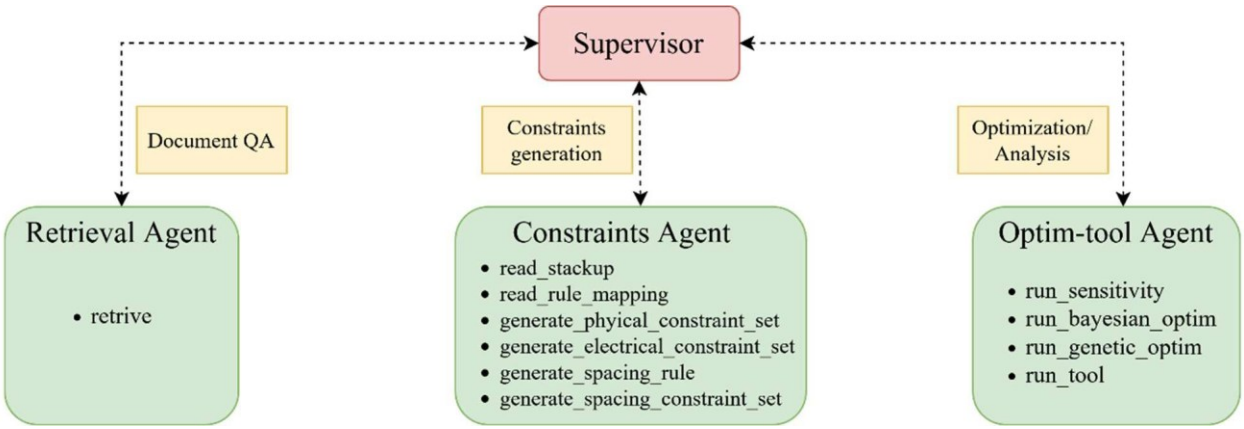

Figure 3 Overview of the multi-agent framework with the tools available to each agent.

Retrieval Augmented Generation (RAG) Agent

Retrieval Augmented Generation (RAG) is an easy, lightweight solution that augments LLMs' knowledge. The ability to include domain-specific or up-to-date information enables LLMs to extend beyond their training knowledge. Furthermore, given the known problem of LLMs generating fictitious information, RAG enables the end user to verify whether the information is correct easily.

RAG consists of two main parts, as shown in Figure 4: the bottom is the preprocessing stage, and the top is the inference stage. In the preprocessing stage, we split the PDG page by page into its constituent parts: text, images, and tables. For multimodal information sources such as images and tables, preprocessing uses two lightweight vision-LLMs (VLLMs) to summarize images and convert table images to textual representations. To summarize the images, we use a llama3-11b-vision model [14], which was good at summarizing images but struggled to convert complex tables effectively. Thus, to handle tables, preprocessing uses IBM's docling model [15], which performs tabular conversion. After summarization and the tabular extraction, the preprocessing reconstructs the page back in a format that resembles the original page in Markdown, making it easier for the LLM to parse and retrieve. The reconstruction also uses docling on the page to infer the structure, which serves as our chunk size for storing embeddings, since the number of tokens per page is typically small in PDGs. The last step of the preprocessing is using an embedding model to generate page embeddings, storing them in a persistent vector database for finding relevant information at inference, and storing the raw information in an object store.

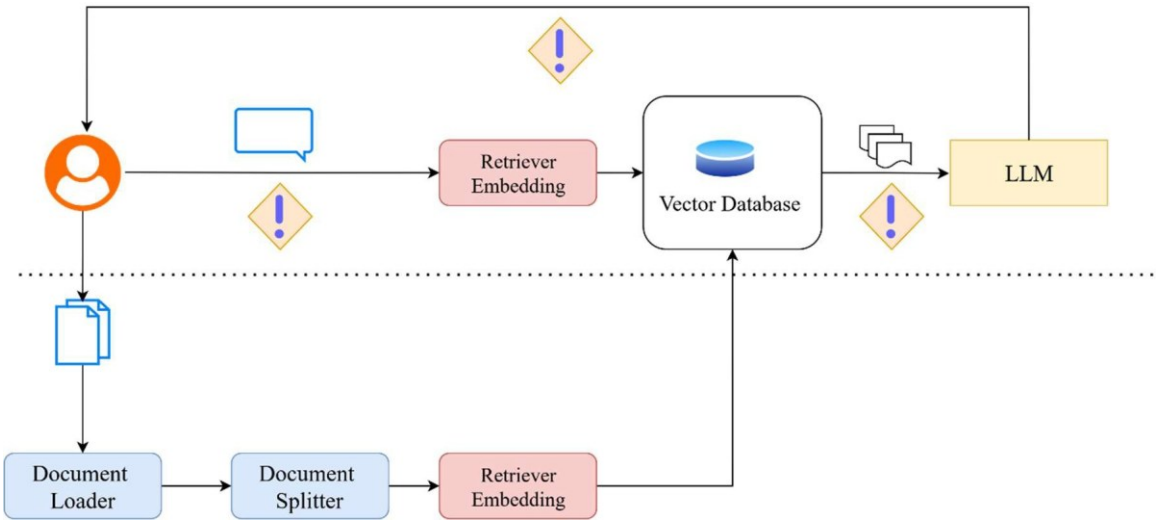

Figure 4. Naïve RAG pipeline with dedicated flows for document processing and handling user queries

During inference, when a user query arrives, the same embedding model converts the query into the embedding space and searches the vector store for document embeddings similar to the user query. The retriever then retrieves the relevant documents from the object store and includes them in the final prompt presented to the LLM.

Though the RAG pipeline reduces the hallucinations, it does not entirely remove them. For this reason, the RAG agent also contains guardrails for the retrieved information and

the final LLM response. These guardrails use LLM-as-a-judge to ensure that the retrieved and final responses are coherent at each stage, given the query and retrieved information. The LLM-as-a-judge assigns a score between 0 and 1 to indicate the quality of the response. The retrieval-based guardrail score is the context relevance, whereas the LLM response guardrail is the answer relevance. The flow combines the two as follows:

$$threshold = 0.25 \times val_{\text{answer\_rel}} + 0.75 \times val_{\text{context\_rel}}$$

As is evident, we prioritize context relevance, primarily because LLMs' responses are coherent when the correct context is present.

The RAG agent is at the core of our multi-agent flow, serving as the primary source of information on vendor design constraints, stackup information, and manufacturing requirements. The RAG agent's information directly responds to the supervisor's query, and the response is either returned to the user or sent to the other two agents.

Constraint Agent

The constraint agent is an LLM with tools for generating routing constraints that span electrical, physical, and spacing domains. The agent is given tools for each domain and aims to generate CSV files that set the constraint in a tool such as Cadence Constraint Compiler. Like the orchestrator/supervisor, the agent here uses ReACT to develop a plan to incorporate design information, given the tools provided to it. It then executes calls to these functions, observes their outputs, and either reruns the flow or ends.

The functions and their utilities that the agent has access to are as follows:

- read_stackup: Reads the stackup information from an Excel file and returns a pandas DataFrame.
- read_rule_mapping: Reads in the required mapping table for the rules, prints it to the console, and returns it as a JSON dictionary as well.
- generate_phyical_constraint_set: Generates the physical constraint set for a given interface (PCIe/DDR/rest) based on the provided stackup.
- generate_electrical_constraint_set: Generate the electrical constraint set using the timing requirements between nets for length matching and time/length based on material, which is from user input.
- generate_spacing_rule: Generates a JSON representation of the spacing rule, including the spacing rule name, rule type, and values for each routing layer, based on the provided user input.
- generate_spacing_constraint_set: Generates the spacing constraint set from the rule's JSON representation.

These functions enable the agent to generate constraints for the different interfaces for a high-speed board. The rules essentially depend on the mapping table, which the user has to generate. Though there are other constraints, such as class-to-class spacing, they rely on the provided set of rules.

Optim-Flow Agent

LLMs are good at many things; however, their use for direct optimization tasks is unclear. In Section 5, we show that LLMs often get stuck in local minima when optimizing problems directly and struggle to reach the optimal solution. In contrast, optimization algorithms such as Bayesian Optimization (BO) find better solutions for a given objective function.

For the aforementioned reasons, the optim-flow agent uses human-crafted optimization algorithms under the hood across numerous domains related to SI/PI and system design. However, in many problems, the user does not know which parameters to optimize; thus, we provide the agent with tools for sensitivity analysis to identify which parameters contribute to the objective. After identifying the relevant parameters, the agent can use numerous algorithms available to it, such as BO and Genetic, to tune the objective's parameters, enabling it to apply to a wide range of tasks.

To ensure a universal interface, the agent requires a user-specified configuration file that specifies the objective function and the relevant parameters. The file specifies the function and final score, which together form the objective. The function here enables the agent to load in different tasks directly, such as calling a SerDes to obtain a BER plot or loading a tool like Simberian Simbeor with templates for structures. The score calculation for the objective is automatically run as part of the function, abstracting it from the agent. However, the score must be user-defined to get relevant results. Depending on the configuration file, the agent either executes a sensitivity analysis or an optimization flow. In this manner, the agent's job is to execute the optimization or sensitivity analysis for a given problem and to explore the use of different algorithms for each task.

Multi-Agent Flow

In this subsection, we present the different flows possible with the framework. In all of these flows, the supervisor agent is the primary interface between the user and the sub-agent, communicating with the user and executing user requests, so it should be a capable model. To confirm that the supervisor executes correctly, the supervisor generates a plan for the user to review and modify before execution.

Figure 2 presents the simplest flows where the supervisor calls a single sub-agent (shown in green/dashed) for a task: for document QA, use the retrieval agent; for generating constraints, use the constraint agent; and for optimization/analysis, use the optim-tool agent.

For complicated flows where the data source is unclear or involves coupling multiple tasks, Figure 5 shows the overall flow. For a given user prompt, the supervisor tries to determine whether it clearly defines the task parameters or requires any missing information. If any information is missing, the supervisor tries to retrieve it using the retrieval agent (step 1). Upon retrieving the necessary information, the supervisor reviews the available data to check if it is ready for the next stage. If the user requests optimal values (step 2), the supervisor uses the optim-tool agent; otherwise, it proceeds to the

constraint agent and generates the relevant constraints (step 3). Further, the supervisor can start from optimization if the user already has the relevant problem defined.

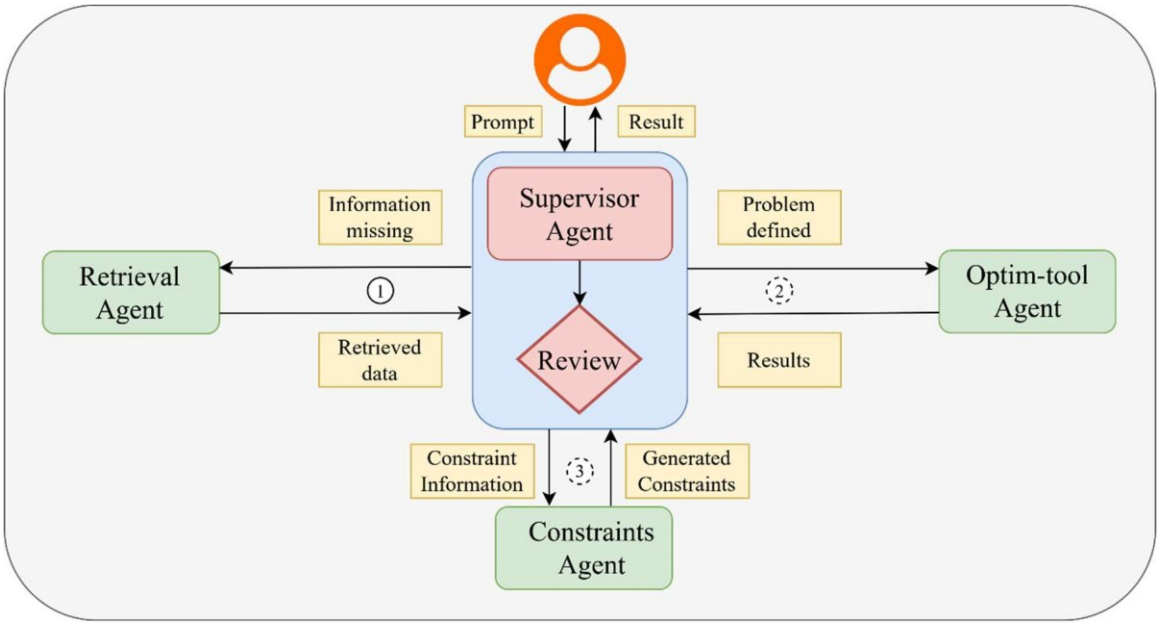

Figure 5. Flow retrieves the information and passes it back to the supervisor, before going to either optim-tool or constraint agent. The optional steps are shown with dashed circles.

Depending on the user's request, the supervisor provides the relevant information. We note that, for flows to proceed via the optim-tool, the configuration file and the structure template must be available in advance.

Design Considerations

While designing the multi-agent flow, we kept several design constraints in mind. Primarily, the full flow runs offline, i.e., data such as PDGs, design files, etc., stays on-premises. The advantage of this is twofold: when newer LLMs become available, it is possible to swap out the current ones without vendor lock-in, and the second being data compliance. To ensure the on-premises aspect, we host the LLM on a network that blocks external connections.

The network access, coupled with our tool chain's Windows dependence, effectively rules out protocols such as the model context protocol (MCP), which aim to make LLMs agentic by providing a unifying interface that specifies the tools available to the model. Thus, we register tools with the LLM on both the LLM system and the compute node and tunnel the responses between them to ensure seamless communication.

Experiments and Results

This section presents the results using our proposed flow. We first demonstrate the supervisor's ability to develop relevant plans for different scenarios. We then walk through the performance of each sub-agent on their specific task: Document QA for the

retrieval agent, compileable constraints for the constraint generation agent, and optimization and sensitivity results for the optim-tool.

4.1. Routing/Planning User Prompts

Figure 6 shows the result of using a Llama-3 model as a supervisor. We prompt the supervisor to generate a spacing constraint for the rule "M2H_S2H" in the DDR routing region, based on our PDG. Given the task, the supervisor agent breaks down the facts present in the task and the facts to look up and derive. Then, based on its thought process, the model develops a plan to look up the DDR section and find the M2H_S2H rule in it (plan steps 1-5). Once the supervisor has all the relevant information, it plans to generate the constraints and verify them (plan steps 6-8). It is clear from this Figure that the supervisor can successfully identify the missing information and create a plan to retrieve it using the retrieval agent. Then, depending on the secondary/tertiary task, it adds it to its plan and calls the relevant agent to execute the task.

Prompt: Can you generate the spacing constraint set for the rule M2H_S2H for DDR?

Initial Plan:
IIere are the facts I know and the plan of action that I will follow to solve the task:
```
### 1.1. Facts given in the task
- The task is to generate the spacing constraint set for the rule M2H_S2H from the DDR section.

### 1.2. Facts to look up
- The DDR section: This could be a reference to a specific document or a section within a document.
  - Where to find it: The document or section where the DDR is located.
  - What to look for: The rule M2H_S2H and its description.
- The rule M2H_S2H: This could be a specific rule within the DDR section.
  - Where to find it: The DDR section or a document that describes the rules.
  - What to look for: The definition of the rule M2H_S2H and its requirements.
- Spacing constraint set: This could be a specific set of constraints related to the rule M2H_S2H.
  - Where to find it: The DDR section or a document that describes the constraints.
  - What to look for: The specific constraints related to the rule M2H_S2H.

### 1.3. Facts to derive
- The definition of the rule M2H_S2H: This can be derived by looking up the DDR section and finding the rule M2II_S2II.
- The requirements of the rule M2H_S2H: This can be derived by looking up the DDR section and finding the rule M2II_S2II.
- The specific constraints related to the rule M2H_S2H: This can be derived by looking up the DDR section and finding the rule M2H_S2H.

### 2. Plan
1. Retrieve the DDR section from the document or section where it is located.
2. Find the rule M2H_S2H within the DDR section.
3. Derive the definition of the rule M2II_S2II from its description in the DDR section.
4. Derive the requirements of the rule M2H_S2H from its description in the DDR section.
5. Derive the specific constraints related to the rule M2II_S2II from its description in the DDR section.
6. Generate the spacing constraint set for the rule M2H_S2H based on the derived constraints.
7. Verify the generated spacing constraint set against the requirements of the rule M2H_S2H.
8. Finalize the spacing constraint set for the rule M2H_S2H.
```

Figure 6. The thought and planning process of the supervisor agent to delegate tasks to sub-agents.

Although we do not show it here, after the supervisor presents its plan, the user retains control by accepting, modifying, or rejecting it. Having the user retain this ability provides another layer of redundancy, allowing them to correct plans they might approach differently.

4.2. Doc QA

Here, we present results from a standard LLM without RAG and from our multimodal RAG-enabled agent. We ask the model for information about the length-matching constraints between two sets of nets. As is evident from Figure 7, the no RAG LLM

generates incorrect information, whereas our multimodal retriever responds with the correct information. Furthermore, in our flow, we display the retrieved context alongside the LLM response so the user can verify that the information is correct.

Prompt: What is the length matching rule for RSP to CLK?

Response: In VHDL, the length matching rule for RSP (Reset Set Point) to CLK (Clock) is that RSP should be a multiple of CLK.

(a) Standard LLM

Prompt: What is the length matching rule for RSP to CLK?

Response: +20 ps to -20ps.

(b) RAG-enabled LLM

Figure 7. Comparing a standard LLM with no-RAG versus ours with RAG for getting length matching requirements.

4.3. Constraint Agent Results

Here, we present the results of the constraint generation agent and demonstrate its ability to generate compileable constraints.

Prompt: Can you generate the physical constraint set for DDR using the old_stackup.xlsx?

Code Execution:

```
stack_df= read_stackup(file_name='old_stackup.xlsx')
print("Stackup read successfully:", stack_df.empty)
print("Stackup shape:", stack_df.shape)
generate_physical_constraint_set(stack_df, rule_set='DDR')
final_answer(generate_physical_constraint_set(stack_df, rule_set='DDR'))
```

Execution Log:

Stack read successfully: False
Stackup shape: (13,33)

Out: Final Answer: True

(a)

```
PhysicalRule,PCS_DDR,,,,,,,,,,,,,,,,,,,,,,,,
,Revision,1,,,,,,,,,,,,,,,,,,,,,,,,
,Units,mil,,,,,,,,,,,,,,,,,,,,,,,,
,Interface, DDR,,,,,,,,,,,,,,,,,,,,,,,,
,Created by,Automated Generation,,,,,,,,,,,,,,,,,,,,,,,,
,Description,All Layers - PCS_DDR,,,,,,,,,,,,,,,,,,,,,,,,
Header,Layer Index,Rule=PCS_DDR,,,,,,,,,,,,,,,,,,,,,,,,
,,MIN_LINE_WIDTH,MIN_NECK_WIDTH,DIFFP_PRIMARY_GAP,DIFFP_NECK_GAP,DIFFP_MIN_SPACE,MAXIMUM_NECK_LENGTH,,
Data,1,6,4,6,4,6,500,,,,,,,,,,,,,,,,,,,,
,3,5.3,3.5,5.7,4,5.7,500,,,,,,,,,,,,,,,,,,,,
,5,5.3,3.5,5.7,4,5.7,500,,,,,,,,,,,,,,,,,,,,
,7,5.3,3.5,5.7,4,5.7,500,,,,,,,,,,,,,,,,,,,,
,9,5.3,3.5,5.7,4,5.7,500,,,,,,,,,,,,,,,,,,,,
,14,5.3,3.5,5.7,4,5.7,500,,,,,,,,,,,,,,,,,,,,
,16,5.3,3.5,5.7,4,5.7,500,,,,,,,,,,,,,,,,,,,,
,18,5.3,3.5,5.7,4,5.7,500,,,,,,,,,,,,,,,,,,,,
,20,5.3,3.5,5.7,4,5.7,500,,,,,,,,,,,,,,,,,,,,
End,,,,,,,,,,,,,,,,,,,,,,,,,
```

(b)

Figure 8. Prompt presented to the LLM to generate constraints for a Physical CSet and the generated constraint for Cadence ACC.

Figure 6 (a) shows the user prompt and the LLM's execution of the relevant functions, and Figure 6 (b) shows the generated PCS constraint for DDR based on the provided stackup.

4.4. Optimization

These results focus on the direct use of LLMs as an optimizer as a motivating factor, and then on the proposed optim-agent flow that enables the agent to apply sensitivity.

4.4.1 Optimization by Prompting on SerDes vs BO

Before adopting the agentic approach to optimization, we wanted to evaluate whether the LLM could drive optimization on its own. To evaluate this flow, we check whether the LLM can optimize a simple SerDes receiver running at 50 Gb/s NRZ using Ansys Electronics Desktop. We use a generic receiver with four parameters to reduce ISI, a single CTLE tap to control the gain, and 3 DFE taps. The main objective was to increase the eye opening by increasing the eye height. We use Llama3.1-3b-Instruct, a substantially sized model, and test both optimization by prompting and an agentic approach that relies on BO. We ensure both approaches use the same initial samples and limit them to 10 turns/iterations.

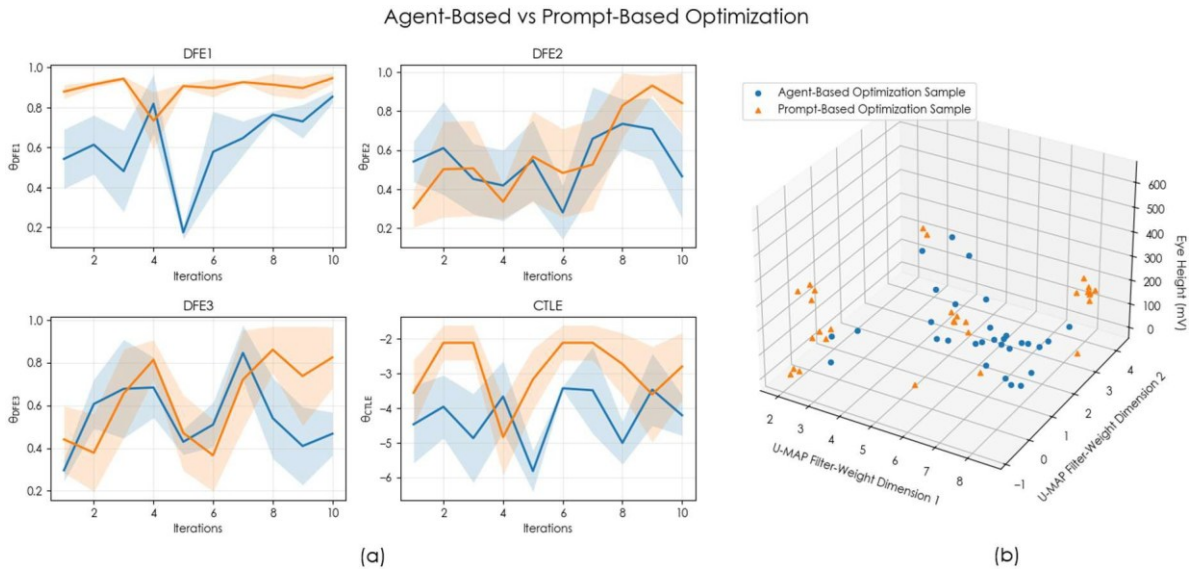

Figure 9. (a) Exploring the tap variation for each tap during every iteration and (b) Exploring the joint projection space of the taps using U-Map.

Figure 9 shows the results using both approaches. Figure 9(a) compares the means of the filter coefficients used for each iteration, whereas Figure 9(b) shows the UMap projection of the coefficients in a joint space. In this experiment, we see that the eye heights are 637 mV and 546 mV for prompt-based and agent-based, respectively. Furthermore, as shown in both figures, the prompt-based approach tends to vary a single filter weight at a time, whereas BO with expected improvement as the acquisition function explores the entire search space. That is why, in Figure 9(b), we see distinct islands, indicating that the LLM recommends changing only values in the neighborhood of an existing value. Though both

approaches achieve a similar eye height, prompting-based optimization in a larger design space would be inefficient.

4.4.2 Agent Driven Optimization Results

Based on the previous results, we leverage the agentic approach to optimization. Here, we present results on an agent that optimizes the taps for a PAM-4 SerDes running at 56 Gb/s. We then show that the agent identifies the relevant parameters for a differential stripline along a constrained route using the sensitivity analysis tool and then optimizes the structure itself.

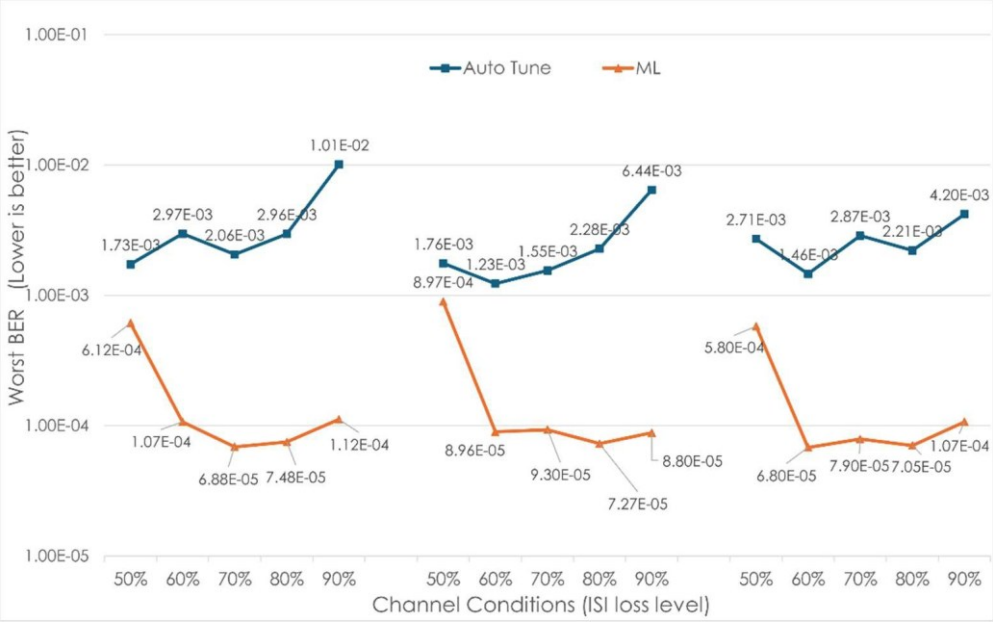

Figure 10. Comparing optimization results for a SerDes running at PAM-4 56 Gb/s across 3 crosstalk configurations (40%, 50%, 60%) and numerous loss conditions for each crosstalk configuration.

Figure 10 shows the result of optimizing a real SerDes connected to the channel and crosstalk emulator. The figure shows three crosstalk configurations at 40, 50, and 60% loss, with channel loss ranging from 50 to 90%. In each case, the receiver has 30 parameters, and the objective is to minimize the worst-case BER. We see that the agent can successfully optimize the SerDes across a range of losses by finding a receiver configuration that consistently yields a lower worst-case BER than the receiver's auto-tune configurations.

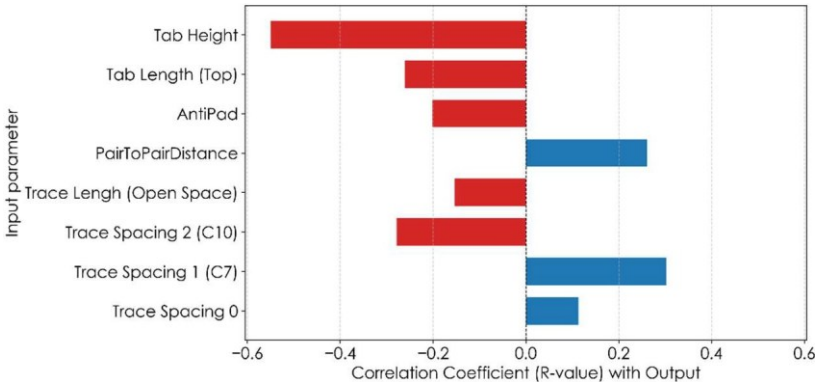

Figure 11. Sensitivity analysis results on a stripline for an end-to-end channel.

The following two results demonstrate the optim-tool agent's ability to first run a sensitivity analysis on an end-to-end routing from pin field to connector, yielding Figure 11, which shows the important parameters that correlate with our performance metric, increasing the gap to the margin specification for a stripline. Then, based on these results, we adjust the configuration file to optimize the structure and increase the gap relative to the margin specification. The combination of sensitivity analysis and optimization yields an optimal stripline, as shown in Figure 12, where the underlying algorithm converges to the optimal solution.

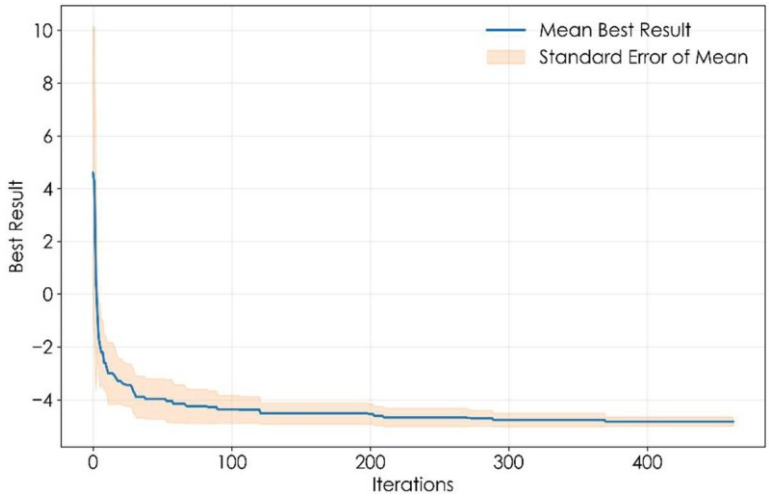

Figure 12. Optimization results on the stripline guided informed by the sensitivity analysis.

Conclusion

This paper presents an agentic approach that spans from a conceptual design specification to routing constraints and optionally enables the user to optimize structures during intermediate steps. The paper presents a lightly coupled supervisor-subagent framework that enables the supervisor to think through a user request, plan it, and call on subagents to execute it. It then shows each agent's performance and how they contribute to achieving the overall goal. It shows that the retrieval agent enables domain-specific knowledge, the constraint-generation agent generates compilable constraints, and the optim-tool agent supports a wide variety of tasks in the design loop and beyond.

Future directions for this work include tightly coupling the subagents so they can communicate directly rather than through the supervisor, and exploring test-time scaling to enable robust reasoning in our flow.

References

[1] D. Zhang, Y. Yu, J. Dong, C. Li and others, "MM-LLMs: Recent advances in multimodal large language models. arXiv 2024." *arXiv preprint arXiv:2401.13601*.

[2] J. Blocklove, S. Garg, R. Karri, and H. Pearce, “Chip-Chat: Challenges and opportunities in conversational hardware design,” in 2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD). IEEE, Sep. 2023, p. 1–6. [Online]. Available: http://dx.doi.org/10.1109/MLCAD58807.2023.10299874

[3] M. Liu, T.-D. Ene, R. Kirby, C. Cheng and others, “ChipNeMo: Domain-adapted LLMs for chip design,” 2024. [Online]. vailable: https://arxiv.org/abs/2311.00176

[4] H. Wu, Z. He, X. Zhang, X. Yao , and others, “ChatEDA: A large language model powered autonomous agent for EDA,” IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 43, no. 10, pp. 3184–3197, 2024.

[5] 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 2024 IEEE LLM Aided Design Workshop (LAD), 2024, pp. 1–5.

[6] Z. Xiao, X. He, H. Wu, B. Yu, and Y. Guo, “EDA-Copilot: A RAG-powered intelligent assistant for EDA tools,” ACM Trans. Des. Autom. Electron. Syst., vol. 30, no. 6, Oct. 2025. [Online]. Available: https://doi.org/10.1145/3715326

[7] D. N. Gadde, K. K. Radhakrishna, V. N. Viswambharan, A. Kumar, and others, “Hey ai, generate me a hardware code! Agentic ai-based hardware design verification,” in 2025 38th SBC/SBMicro/IEEE Symposium on Integrated Circuits and Systems Design (SBCCI). IEEE, Aug. 2025, p. 1–5. [Online]. Available: http://dx.doi.org/10.1109/SBCCI66862.2025.11218681

[8] F. Firouzi, D. Z. Pan, J. Gu, B. Farahani, and others, “ChipMnd: LLMs for agile chip design,” in 2025 IEEE 43rd VLSI Test Symposium (VTS), 2025, pp. 1–10.

[9] A. Ghose, A. B. Kahng, S. Kundu, and Z. Wang, “Orfs-agent: Tool-using agents for chip design optimization,” 2025. [Online]. Available: https://arxiv.org/abs/2506.08332

[10] S. Raschka, “The Big LLM Architecture Comparison.” Ahead of AI. Jul. 19, 2025. [Online]. Available: https://magazine.sebastianraschka.com/p/the-big-llm-architecture-comparison.

[11] D. Testuggine, “A Primer on LLM Post-Training.” PyTorch. Aug. 26, 2025. [Online]. Available: https://pytorch.org/blog/a-primer-on-llm-post-training/

[12] J. Wei, X. Wang, D. Schuurmans, et al., “Chain-of-thought prompting elicits reasoning in large language models,” in Proceedings of the 36th International Conference on Neural Information Processing Systems, New Orleans, LA, USA: Curran Associates Inc., 2022, ISBN: 9781713871088.

[13] S. Yao, J. Zhao, D. Yu, et al., “ReAct: Synergizing reasoning and acting in language models,” in International Conference on Learning Representations (ICLR), 2023.

[14] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, and others, “The Llama 3 Herd of Models,” arXiv preprint arXiv: 2407.21783, 2024.

[15] N. Livathinos, C. Auer, M. Lysak, A. Nassar, and others, “Docling: An Efficient Open-Source Toolkit for AI-driven Document Conversion,” arXiv preprint arXiv: 2501.17887, 2025.

Acknowledgements

This material is based upon work supported by the National Science Foundation under Grant No. CNS 2137283 - Center for Advanced Electronics through Machine Learning (CAEML) and its industry members.