Divergent Thoughts toward One Goal: LLM-based Multi-Agent Collaboration System for Electronic Design Automation

Haoyuan Wu[♠], Haisheng Zheng[♡], Zhuolun He[♠,♣], Bei Yu[♠]

[♠]The Chinese University of Hong Kong, Hong Kong SAR
[♡]Shanghai Artificial Intelligent Laboratory, China
[♣]ChatEDA Tech, China
{hywu24,byu}@cse.cuhk.edu.hk

Abstract

Recently, with the development of tool-calling capabilities in large language models (LLMs), these models have demonstrated significant potential for automating electronic design automation (EDA) flows by interacting with EDA tool APIs via EDA scripts. However, considering the limited understanding of EDA tools, LLMs face challenges in practical scenarios where diverse interfaces of EDA tools exist across different platforms. Additionally, EDA flow automation often involves intricate, long-chain tool-calling processes, increasing the likelihood of errors in intermediate steps. Any errors will lead to the instability and failure of EDA flow automation. To address these challenges, we introduce EDAid, a multi-agent collaboration system where multiple agents harboring divergent thoughts converge towards a common goal, ensuring reliable and successful EDA flow automation. Specifically, each agent is controlled by ChipLlama models, which are expert LLMs fine-tuned for EDA flow automation. Our experiments demonstrate the state-of-the-art (SOTA) performance of our ChipLlama models and validate the effectiveness of our EDAid in the automation of complex EDA flows, showcasing superior performance compared to single-agent systems.

1 Introduction

Electronic design automation (EDA) is indispensable for the design of integrated circuits (ICs). EDA tools are integrated into a complex design flow and utilize programming interfaces to control the design process. EDA platforms such as OpenROAD (Ajayi and Blaauw, 2019) and iEDA (Li et al., 2024), consist of complex procedures with various configurations. Circuit design engineers utilize EDA tools iteratively to fulfill design targets, relying on tailored scripts that manipulate these tools via programming interfaces. However, interacting with EDA tools through scripting (Chen et al., 2001) is often laborious and error-prone. This complexity is further intensified when design teams employ tools from various vendors in the circuit design process.

Large language models (LLMs) (Achiam et al., 2023; Anthropic, 2024; Dubey et al., 2024) have demonstrated profound instruction comprehension, planning, and reasoning capabilities. The potential of LLMs to interact with diverse tools for executing complex tasks has gained increasing recognition (Qin et al., 2023). Researchers have explored the automation of complex EDA flows by interfacing with EDA tools via LLMs (Wu et al., 2024; Liu et al., 2023). Specifically, ChatEDA (Wu et al., 2024) uses LLMs as "brains" of the agent to generate EDA scripts, automating EDA tool utilization, reducing the workload of circuit design engineers, and minimizing errors.

Although LLMs have shown potential in EDA flow automation, significant challenges remain. Firstly, although LLMs excel at understanding natural language, they lack specialized knowledge of EDA tool usage. These tools are designed for specific tasks such as logic synthesis, floorplanning, placement, and routing, each requiring detailed domain-specific knowledge and familiarity with various EDA tool interfaces. To solve these problems, LLMs can be fine-tuned on datasets containing tutorials and EDA scripts specific to EDA tools (Wu et al., 2024). However, this approach presents its problems. Each EDA platform has a unique set of commands and flows that must be mastered for effective use. If LLMs only focus on a specific EDA tool or platform during the instruction tuning process will limit their cross-platform utility, potentially reducing the effectiveness in practical scenarios. Furthermore, EDA flows typically involve a sequence of intermediate steps. A single error in any of these steps can lead to failure in the overall process. This risk is compounded by the probabilistic nature of LLMs, which may pro-

duce varying solutions to the same task. Moreover, errors may occur in intermediate steps during a long-chain tool-calling process, introducing instability into the EDA flow automation.

To address these challenges, we introduce a multi-agent collaboration system, EDAid, designed for EDA flow automation through script generation in response to natural language instructions. This system is characterized by the collaboration of multiple agents, which are powered by the LLMs. As mentioned earlier, EDA flow automation is challenging even for the greatest LLM such as GPT-4 (Achiam et al., 2023). Consequently, we develop ChipLlama models, expert LLMs fine-tuned for EDA flow automation. We specifically focus on improving the understanding of overall EDA flow rather than simple EDA tool usage during the fine-tuning process. We also employ few-shot chain-of-thought (CoT) prompts for each agent to further develop the performance and portability through the in-context learning capability of LLMs. Based on multiple agents, we propose the multi-agent system, EDAid, to ensure stability and avoid erroneous intermediate steps during the long-chain EDA tool calling process. In this system, multiple agents collaborate with divergent thoughts following different few-shot contexts and then make the final decision based on divergent thoughts, working in concert to automate the EDA process. Our EDAid can interpret human instructions, plan EDA tasks, and interface with EDA tools through APIs, serving as a valuable assistant in automating EDA flows and eliminating the need for manual intervention. In summary, our contributions are as follows:

- We develop the ChipLlama-powered agent, collaborating with few-shot CoT prompts, to develop the performance and portability of the single-agent system;
- Propose EDAid, a multi-agent system that collaborates multiple agents including divergent-thoughts agents and a decision-making agent for EDA flow automation;
- Perform extensive evaluations, which demonstrate the SOTA performance of ChipLlama models, the effectiveness of the few-shot CoT prompting, and the superior performance of our EDAid for EDA flow automation.

2 ChipLlama-powered Agent

For an autonomous agent for EDA flow automation, users can provide the EDA task in natural language,

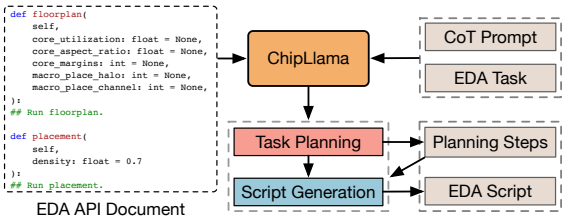

Figure 1: Overview of the ChipLlama-powered agent for task planning and EDA script generation.

and the agent will generate executable scripts to complete the EDA task via an LLM. To guarantee the performance and reliability of the overall flow, we introduce the expert LLM, ChipLlama, as the "brain" of the single-agent system. Furthermore, we apply few-shot CoT prompts to the single-agent system to enhance the reasoning ability for EDA flow automation as shown in Figure 1.

2.1 ChipLlama for EDA Flow Automation

ChipLlama models are expert LLMs fine-tuned based on Llama3 (Dubey et al., 2024) via the hybrid instruction tuning. The capability of the ChipLlama models determines the performance of EDA task-resolving. Previous expert LLMs for EDA flow automation (e.g. AutoMage2 (Wu et al., 2024)) only show reliable performance on a single platform. However, there are various EDA platforms in real industrial scenarios. To address this challenge, we enhance the generalization ability of LLMs by broadening their understanding of the entire EDA flow, beyond mere simple tool usage.

The invocation of EDA tools is dependent on maintaining the correct logical order in contrast to other fields where tool usage may not require a strict logical sequence. This dependency arises from the nature of EDA processes, which consist of interconnected stages such as logic synthesis, floorplanning, placement, and routing. Each stage must be executed sequentially, relying on precise inputs from preceding steps. Therefore, besides domain-specific knowledge for EDA tool usage, LLMs must exhibit advanced logical reasoning and tool manipulation (via code) skills.

Instruction tuning is founded on the principle that by engaging in supervised learning driven by task-specific instructions, LLMs can acquire the skill to adhere to directives for tasks they have not previously encountered. This facilitates the application of LLMs to EDA tasks utilizing datasets from domains beyond EDA. Instruction datasets from various fields provide a wealth of directives,

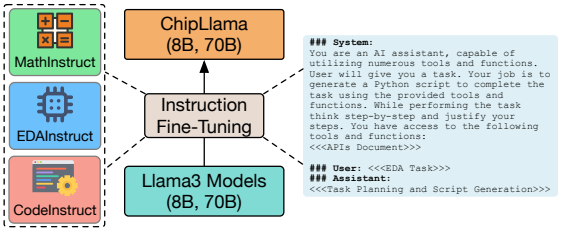

Figure 2: Overview of hybrid instruction tuning.

promoting the ability of models to develop versatile problem-solving strategies by correlating inputs with outputs. Consequently, as illustrated in Figure 2, we utilize hybrid instruction tuning for ChipLlama models, which integrates three specially curated datasets including MathInstruct (Yu et al., 2023), CodeInstruct (Wei et al., 2023), and EDAInstruct (Wu et al., 2024). Hybrid instruction tuning fosters LLMs' deep understanding of sophisticated EDA flow automation, which can be generalized to various EDA platforms and enhance the performance and reliability applying to a single EDA platform. We show more details of hybrid instruction tuning in Appendix A.

2.2 Few-shot CoT Prompting

Backend design involves complex procedures to interact with various EDA tools. For example, achieving timing closure has to optimize cell placement, clock trees, and signal routing iteratively until satisfying performance is obtained. In this sense, it is difficult for LLMs to generate a script that finishes the job directly (Wu et al., 2024). The enhancement for LLMs in such script generation can be achieved through the application of CoT prompting (Wei et al., 2022b). As illustrated in Figure 1, after receiving user instruction and API document, ChipLlama models start to plan how to complete the task in several steps following the CoT prompt and then generate the corresponding script according to the planning steps.

Let's first focus on the standard prompt for EDA task-solving without the task planning phase. For standard prompting, the objective is to maximize the probability of the script $\mathcal{A}$ given an EDA task $\mathcal{Q}$, a prompt $\mathcal{T}$, and the probabilistic LLM $p_{\mathcal{L}}$ representing the utilized ChipLlama model. The probability is expressed as:

$$p(\mathcal{A}|\mathcal{Q},\mathcal{T}) = \prod_{i=0}^{|\mathcal{A}|} p_{\mathcal{L}}(a_i|\mathcal{Q},\mathcal{T},\mathcal{A}_{<i}), \quad (1)$$

where $\mathcal{A}_{<i} = \{a_1, a_2, \cdots a_{i-1}\}$, a_i represents the i-th token and $|\mathcal{A}|$ denotes the length of the EDA script $\mathcal{A}$.

```
### System: You are an AI assistant, capable of utilizing numerous tools
and functions. User will give you a task. Your job is to generate a Python
script to complete the task using the provided tools and functions. While
performing the task think step-by-step and justify your steps. You have
access to the following tools and functions:
<<<APIs Document>>>

### User: <<<EDA Task 1>>>
### Assistant: <<<Solution 1>>>

### User: <<<EDA Task 2>>>
### Assistant: <<<Solution 2>>>

### User:
For the "aes" circuit, I want to run the steps from setup to detailed
routing on the platform "asap7"?
Let's first describe and explain what the task is asking. Then, analyze how
to complete the task step by step using the provided tools and functions.
Finally, generate the Python script according to your analysis.
### Assistant:
```

Figure 3: Few-shot CoT prompt template utilized in ChipLlama-powered agent.

Zero-shot CoT prompting is a straightforward concept that incorporates a sequence of task planning steps into the initial prompt. In this scenario, p_{L_0} processes $\mathcal{Q}$ and $\mathcal{T}$ to generate a models of planning steps $\mathcal{C}$ and then generate the EDA script $\mathcal{A}$. In addressing $\mathcal{Q}$, articulated in natural language, the expert LLM from the ChipLlama models dissects it into a sequential set of steps using $\mathcal{V}$ as per the guidelines outlined in the API document within $\mathcal{T}$. This decomposition facilitates streamlined handling through the utilization of EDA tools. Furthermore, meticulous determination of parameters required at each step occurs during the task planning phase, minimizing the potential for erroneous parameter usage in EDA script generation. Following the task planning phase, structured steps $\mathcal{C}$ are formulated, enhancing the efficient orchestration of the intricate EDA task. Each step is executable through the corresponding APIs of the EDA tools. Subsequently, LLMs can formulate the script $\mathcal{A}$ to invoke these APIs for automating the EDA flow. Therefore, Equation (1) can be modified to:

$$p(\mathcal{A} \mid \mathcal{Q},\mathcal{T}) = p(\mathcal{A} \mid \mathcal{Q},\mathcal{T},\mathcal{C})p(\mathcal{C} \mid \mathcal{Q},\mathcal{T}), \quad (2)$$

where $p(\mathcal{A} \mid \mathcal{Q},\mathcal{T},\mathcal{C})$ and $p(\mathcal{C} \mid \mathcal{Q},\mathcal{T})$ are defined as follows:

$$p(\mathcal{C} \mid \mathcal{Q},\mathcal{T}) = \prod_{i=0}^{|\mathcal{C}|} p_{\mathcal{L}}(c_i \mid \mathcal{Q},\mathcal{T},\mathcal{C}_{<i}), \quad (3)$$

$$p(\mathcal{A} \mid \mathcal{Q},\mathcal{T},\mathcal{C}) = \prod_{i=0}^{|\mathcal{A}|} p_{\mathcal{L}}(a_i \mid \mathcal{Q},\mathcal{T},\mathcal{C},\mathcal{A}_{<i}). \quad (4)$$

Here, $\mathcal{C}_{<i} = \{c_1, c_2, \cdots c_{i-1}\}$. c_i and $|\mathcal{C}|$ indicate the i-th token and the length of the task planning steps, respectively. Zero-shot CoT prompt is provided to guide LLMs in generating task planning steps $\mathcal{C}$ before generating the EDA script $\mathcal{A}$.

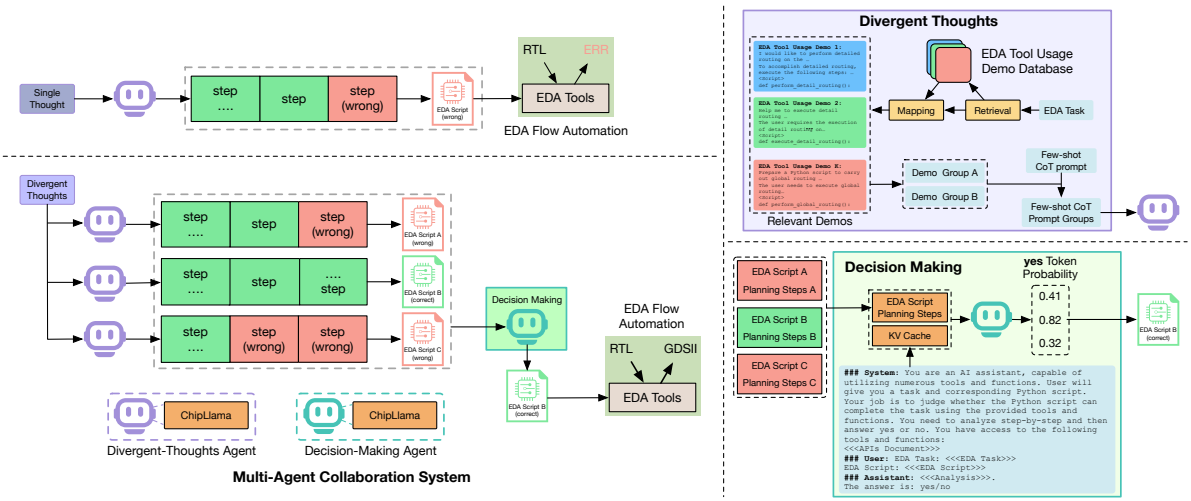

Figure 4: Overview of EDAid, the multi-agent collaboration system. Given an EDA task, multiple agents (including divergent-thoughts agents (role R_0) and a decision-making agent (role R_1)) collaborate to generate the EDA script. Finally, the generated EDA script will automate the EDA flow interfacing the EDA tools via APIs.

Few-shot CoT prompting merges the paradigms of in-context learning with zero-shot CoT prompting to enhance performance on complex EDA tasks that necessitate planning guidelines for resolution. In few-shot CoT scenario, $\mathcal{T} = (\mathcal{Q}_i, \mathcal{C}_i, \mathcal{A}_i)_{i=1}^{\mathcal{N}}$, which consists of $\mathcal{N}$ instances of $(\mathcal{Q}, \mathcal{C}, \mathcal{A})$ tuple. For an instance $(\mathcal{Q}, \mathcal{C}, \mathcal{A})$, this instance serves as a guide for resolving the EDA task. Initially, it directs LLMs to decompose the EDA task $\mathcal{Q}$ according to the API document. Following the decomposition steps, encompassing logic synthesis, floorplan creation, placement, clock tree synthesis (CTS), routing, and other relevant processes, is generated within the context of $\mathcal{C}$. Ultimately, the instance guides LLMs on generating the EDA script $\mathcal{A}$ based on the decomposed steps $\mathcal{C}$. As shown in Figure 3, a few-shot CoT prompt is provided to guide LLMs on how to generate task planning steps $\mathcal{C}$ and the EDA script $\mathcal{A}$ following previous EDA tool usage demos.

LLMs can acquire the skills to plan for an EDA task and generate the corresponding script by learning from a set of $\mathcal{N}$ instances in a few-shot CoT prompt $\mathcal{T}$. Moreover, consider the scenario where a design team has acquired a brand new EDA tool. By providing ChipLlama models with brand new EDA APIs, they can still draw knowledge for EDA flow automation from few-shot CoT prompts.

3 Multi-Agent Collaboration System

Automating EDA flows within complex real-world scenarios poses a formidable challenge for single-agent systems (shown in Figure 6). These complexities arise from the multidimensional nature of EDA projects, which demand technical expertise and the need for a long-chain EDA tool-calling process. As shown in Figure 4, the single-agent system will make mistakes in the intermediate steps during the complex EDA task planning process, leading to failure in EDA flow automation. In response, we introduce EDAid, a collaborative multi-agent system designed for EDA flow automation. Unlike the single-agent system utilizing the vanilla CoT, this system features multiple agents operating in synergy with self-consistency to resolve more complex EDA tasks. In the following subsections, we will elucidate the mechanisms within the multi-agent system, which coordinate the efforts of multiple LLM-powered agents to automate the EDA flow.

3.1 Multi-Agent System Definition

For clear clarification, we conceptualize the operational environment of the multi-agent system as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ represents the nodes corresponding to the agents, and $\mathcal{E}$ signifies the edges that delineate the communicative links among the agents.

Agent Definition. Each agent $i \in \mathcal{V}$ is defined by a tuple $\mathcal{V}_i = (L_i, R_i)$. Here, L_i denotes the specific instances of LLMs employed by the agent, including their types and configurations, and the different prompt designs for different LLMs. R_i designates the agent's role, which determines its duties and imparts a defined purpose and direction, thereby steering its actions and interactions. As shown in Figure 4, two distinct roles are identified.

Specifically, divergent-thoughts agents (role R_0) are tasked with comprehending user inquiries and generating scripts for EDA flow automation. The decision-making agent (role R_1) is the decision-maker who selects the optimal solution from various solutions to resolve the EDA task.

Connection and Message Definition. Each edge $e_{ij} \in \mathcal{E}$ establishes a communicative link between agent $\mathcal{V}_i$ and $\mathcal{V}_j$, facilitating message exchange. A message m encapsulates the content that includes the task plan and the relative EDA script, which can be transmitted from agent $\mathcal{V}_i$ to $\mathcal{V}_j$ via the established channel e_{ij}.

3.2 Divergent Thoughts

Considering the superior performance of in-context learning, most tasks in various domains can be resolved by LLMs in one go. However, LLMs exhibit errors in EDA flow automation considering their multiple complex planning steps. A notable characteristic of human cognition is the diversity of thought processes. Similarly, multiple planning pathways can be employed to address EDA tasks. Single-agent systems are not infallible in task planning, especially for scenarios that require long-chain tool-calling processes. As depicted in Figure 4, a single-agent system may occasionally pursue incorrect planning steps or commit errors within the task planning process. Specifically, a single-agent system might engage in placement without preceding floorplanning execution, or it could employ parameters tailored for the CTS stage at the placement phase. Such flawed planning can not converge on the correct EDA script (shown in Figure 6). Consequently, multiple planning pathways searching via multiple divergent-thoughts agents is necessary.

According to Equation (3), planning steps $\mathcal{C}$ are contingent upon the prompt $\mathcal{T}$ when employing greedy decoding in CoT prompting. By supplying prompts derived from distinct distributions, we can engender a variety of planning steps $\mathcal{C}$, which facilitates the generation of answers $\mathcal{A}$ from divergent-thoughts agents that reflect divergent thinking. In the context of few-shot CoT prompts, where $\mathcal{T} = (\mathcal{Q}_i, \mathcal{C}_i, \mathcal{A}_i)_{i=1}^{\mathcal{N}}$, employing different permutations of $(\mathcal{Q}, \mathcal{C}, \mathcal{A})$ tuples can lead to the generation of $\mathcal{O} = \{\mathcal{O}_1, ..., \mathcal{O}_i\}$ that embody divergent thoughts. The divergent thoughts process consists of two core components: relevant EDA tool usage demos retrieval and generation of different few-shot prompts with varied $(\mathcal{Q}, \mathcal{C}, \mathcal{A})$ tuples.

Relevant EDA Tool Usage Demos Retrieval. As shown in Figure 4, the EDA tool usage demo database contains numerous instances, each of which includes a $(\mathcal{Q}, \mathcal{C}, \mathcal{A})$ tuple. The embedding model encodes each EDA task $\mathcal{Q}$ into a vector, creating an EDA task vector database. For a new EDA task, the cosine similarity is computed between its embedding and all vectors in the vector database. The Top-K most similar tasks are identified, and their corresponding IDs are mapped back to the demo database to retrieve the K most relevant instances with their respective $(\mathcal{Q}, \mathcal{C}, \mathcal{A})$ tuples.

Different Few-shot Prompts Generation. As shown in Figure 4, several instances are randomly chosen from the retrieved pool of K relevant demos to create a demo group. This random selection process is iterated to produce multiple demo groups, such as Group A, Group B, Group C, etc. Each demo group is concatenated with the few-shot prompt template (displayed in Figure 3), yielding several few-shot prompt groups with different prompts $\mathcal{T} = (\mathcal{Q}_i, \mathcal{C}_i, \mathcal{A}_i)_{i=1}^{\mathcal{N}}$. Upon inputting these differing few-shot prompts into divergent-thoughts agents, they generate various outputs $\mathcal{O}$ according to the given prompts, thus realizing the objective of divergent thoughts.

3.3 Decision Making

Multiple divergent-thoughts agents $\mathcal{V}_i$ generate diverse outcomes $\mathcal{O}_i = (\mathcal{C}_i, \mathcal{A}_i)$, reflecting divergent thoughts with different few-shot prompts. These outcomes $\mathcal{O}_i \in \mathcal{O}$ are encapsulated within a message $m = \{\mathcal{O}_1, ..., \mathcal{O}_i\}$ and subsequently dispatched to the decision making agent $\mathcal{V}_j$ with role R_1. Upon receipt, $\mathcal{V}_j$ start to perform a multiple choice selection with various $\mathcal{O}_i$ as choices, and determines which $\mathcal{O}_i$ should be selected as the representative output of the multi-agent system. As depicted in Figure 4, the generated EDA script will automate the EDA flow interfacing with EDA tools.

As illustrated in Figure 4, we combine the EDA task and prompt into a complete input and feed it into the decision-making agent for multiple choice selection. Specifically, we ask the decision-making agent to analyze and judge whether the generated EDA scripts can solve the EDA task. Next, we calculate the probability of selecting the **yes** token from $\{\textbf{yes}, \textbf{no}\}$ at the end of the answer for each candidate EDA script as shown in Figure 4. Finally, we select the candidate answer with the highest **yes** token probability as the representative output of the EDAid. Moreover, considering that the input

Simple Flow Calls
Case 1: Please show me how to complete the design flow for the "aes" circuit on the platform "asap7".

Case 2: Can you help me to evaluate how channel values impact the performance after the clock tree synthesis stage?

Complex Flow Calls
Case 1: Try to find out the smallest valid clock period for the design "leon" on "asap7" platform. Note that a clock period is valid only if the "wns" metric at the final stage is non negative.

Case 2: I want to conduct a grid search of the design space including varying floorplan and placement parameters for my design called "how" on the platform "nangate45". Can you do that?

Parameter Tuner Calls
Case 1: For the design "aes" on "nangate45" platform, please write me a script to optimize ppa metrics using the parameter tuning method while setting clock period to 5.

Case 2: I want to perform dse for my design "asadsf" on nangate45 with default placement and synthesis parameters. Using performance and area as evaluation metrics for parameter tuning.

Figure 5: Examples of evaluation benchmarks.

part is the same for different candidate answers, we store the system prompt in the KV Cache to avoid redundant computation.

4 Experiments

4.1 Experiments Setting

We utilize a comprehensive evaluation benchmark ChatEDA-bench (Wu et al., 2024), which comprises 50 distinct tasks with target APIs from OpenROAD (Ajayi and Blaauw, 2019), to evaluate the performance of our EDAid and ChipLlama models. Moreover, we also design a benchmark, iEDA-bench, based on iEDA (Li et al., 2024) comprising 50 distinct tasks to evaluate the generalization to any EDA tools on different platforms of ChipLlama. ChatEDA-bench and iEDA-bench use the accuracy of the generated EDA script as the evaluation metric. Specifically, accuracy is related to the successful EDA flow automated through the correct generated EDA script.

4.2 Examples of Evaluation Benchmarks

As illustrated in Figure 5, we show some examples of our evaluation benchmarks including ChatEDA-bench (Wu et al., 2024) and iEDA-bench, both of which are comprehensive evaluation benchmarks comprising 50 distinct tasks including three distinct categories: simple flow calls (30%), complex flow calls (30%), and parameter flow calls (40%).

Simple Flow Calls. This task requires the successful execution of the whole process, including evaluation. These cases test the fundamental application of LLMs in EDA flow automation.

Complex Flow Calls. This task requires a higher proficiency in EDA tool usage, including traversing parameters, which examines logical reasoning and understanding of each argument of EDA APIs.

System	Powered LLM	ChatEDA-bench Acc.	iEDA-bench Acc.
ChatEDA	GPT-3.5$^\diamond$	28%	30%
ChatEDA	GPT-4$^\diamond$	62%	70%
ChatEDA	AutoMage-70B$^\diamond$	74%	-
ChatEDA	AutoMage2-70B$^\diamond$	82%	-
EDAid	ChipLlama-8B	88%	84%
EDAid	ChipLlama-70B	**100%**	**100%**

$^\diamond$ The accuracy values of GPT-3.5, GPT-4 and AutoMage models on the ChatEDA-bench are directly cited from the ChatEDA (Wu et al., 2024). Moreover, we can only evaluate AutoMage models on the ChatEDA-bench due to the unavailability of closed-source models.

Table 1: The main results of EDA script generation on ChatEDA-bench (Wu et al., 2024) and iEDA-bench.

Parameter Tuner Calls. This task requires agent systems to provide a parameter-tuning solution, which is a vital step for EDA considering the complexity of the entire process.

4.3 Implementation Details

We employ QLoRA (Dettmers et al., 2024) for the hybrid instruction tuning of ChipLlama models based on Llama3 (Dubey et al., 2024) models. This involves adopting a constant learning rate schedule with a warm-up ratio of 0.03, utilizing the paged AdamW optimizer (Dettmers et al., 2021) with a learning rate of 1×10^{-4}, no weight decay, a batch size of 128, and a sequence length of 4096 tokens. The models are fine-tuned for 1 epoch on 16×A100 GPUs with 80G memory each. After hybrid instruction tuning, we obtain ChipLlama-8B and ChipLlama-70B, two expert LLMs for EDA flow automation. For clarity and differentiation, the single-agent system refers to the divergent-thoughts agent (role R_0) in EDAid, while the multi-agent system denotes our EDAid.

4.4 Main Evaluation Results

In this study, we integrate LLMs into our EDAid to conduct a comprehensive evaluation. According to (Wu et al., 2024), AutoMage models serve as "brains" of the ChatEDA system to execute EDA tasks. Concurrently, we incorporate GPT-3.5 (Brown et al., 2020) and GPT-4 (Achiam et al., 2023) into the single-agent system (ChatEDA) through their official APIs. Importantly, only open-source models are utilized within our multi-agent system, as the decision-making process requires calculating the output logits of LLMs.

As demonstrated in Table 1, our multi-agent system, EDAid, powered by ChipLlama-70B, achieves the SOTA performance over all other previous SOTA LLM-powered systems, establishing a significant margin. To explore the generalization of our

System	Base LLM	Hybrid Instruction Tuning	ChatEDA-bench Acc.	iEDA-bench Acc.
Single-Agent	Llama3-8B	✗	**78%**	50%
Single-Agent		✓	**78%**	**76%**
Single-Agent	Llama3-70B	✗	88%	74%
Single-Agent		✓	**94%**	**96%**

Table 2: Ablation study on hybrid instruction tuning.

System	Powered LLM	ChatEDA-bench Acc.		iEDA-bench Acc.	
		zero-shot	few-shot	zero-shot	few-shot
Single-Agent	GPT-3.5	28%	56%	30%	50%
Single-Agent	GPT-4	62%	82%	70%	84%
Single-Agent	ChipLlama-8B	74%	78%	64%	76%
Single-Agent	ChipLlama-70B	**90%**	**94%**	**90%**	**96%**

Table 3: Ablation study on few-shot prompting and powered LLMs of the single-agent system.

System	Powered LLM	ChatEDA-bench Acc.	iEDA-bench Acc.
Single-Agent	ChipLlama-8B	78%	76%
Multi-Agent		**88%**	**84%**
Single-Agent	ChipLlama-70B	94%	96%
Multi-Agent		**100%**	**100%**

Table 4: Ablation study on single/multi-agent systems powered by different LLMs.

EDAid on utilizing EDA tools, we also evaluate it on iEDA-bench, which requires LLMs to use EDA tools from the iEDA platform. Similarly, EDAid powered by ChipLlama-70B also demonstrates outstanding accuracy. Our multi-agent collaboration system manifests its potential in automating the EDA flow by interfacing with diverse EDA tools from various vendors.

4.5 Ablation Studies

In the following section, we conduct four ablation studies to further illustrate the effectiveness of our ChipLlama models and our multi-agent collaboration system, EDAid.

Hybrid Instruction Tuning. We compare hybrid instruction tuning and simple EDA-domain instruction tuning and show the results in few-shot scenarios in Table 2. These results underscore the robust generalization capabilities of ChipLlama models after hybrid instruction tuning in utilizing EDA tools across different platforms (e.g. iEDA), notwithstanding its initial training exclusively on EDA tools on the single EDA platform (OpenROAD). Moreover, hybrid instruction tuning also achieves improvement on the ChatEDA-bench, which demonstrates that this strategy can also help LLMs to improve their complex reasoning and long-chain tool-calling capabilities.

Powered LLMs of Single-agent System. Considering the powered LLM of each agent in EDAid is vital to reliable EDA flow automation, we test various LLMs by serving them as the controller of the single agent system. As shown in Table 3, our ChipLlama-70B achieves significant improvements compared to GPT-4 in zero-shot and few-shot scenarios across different EDA platforms. Notably, ChipLlama-8B also achieves comparable performance to GPT-4, which is impressive considering its model parameters.

Few-shot CoT Prompts. We verify the performance with zero-shot prompts and few-shot prompts and the results are shown in Table 3. We can observe that GPT models and ChipLlama models demonstrate their capabilities in few-shot learning, which demonstrates the advantages offered by few-shot prompts compared to zero-shot prompts. Meanwhile, it is worth noting that few-shot CoT prompting is also of great benefit for models' portability across different EDA platforms.

Multiple Agents Collaboration. To assess the efficiency of multi-agent collaboration, we utilize the ChipLlama models to control the agents in our multi-agent collaboration system to perform EDA script generation. Table 4 illustrates that the collaboration of multiple agents brings improvement over the single agent for ChipLlama models. This enhancement underscores the capacity of our multi-agent collaboration system to provide dependable assistance in automating the EDA flow.

5 Case Studies

In this section, we provide case studies about agents of EDAid, including divergent thoughts agent (role R_0) and decision-making agent (role R_1), to demonstrate the core capabilities that enable EDAid to function effectively.

As shown in Figure 6, the divergent-thoughts agent can generate correct task planning pathways and the corresponding EDA script that can automate the EDA flow successfully. However, it is common that the divergent-thoughts agent makes mistakes in intermediate steps during the EDA flow automation, which requires long-chain tool-calling capability. Specifically, the "global_route" method does not have a parameter called "macro_place_channel". The "macro_place_channel" is a parameter for the

Figure 6: Divergent thoughts. Right: the divergent-thoughts agent (role R_0) generates correct task planning pathways and EDA script. Left: the divergent-thoughts agent (role R_0) makes mistakes in intermediate steps.

"floorplan" method. Although most of the task planning pathways are correct, these single errors can still lead to failure in the overall process.

As illustrated in Figure 7, the decision-making agent can accurately identify and fix errors of given EDA scripts according to the given EDA tasks. This capability of error-correct provides a solid base for our decision-making process in our multi-agent collaboration system. Specifically, the decision-making agent can select the correct EDA script from divergent thoughts (shown in Figure 6), which guarantees the performance and effectiveness of our multi-agent system, EDAid.

Moreover, we also provide more case studies in Appendix C.

6 Related Works

Chain-of-Thought. The CoT paradigm (Wei et al., 2022b) encourages LLMs to break down complex problems into several intermediate steps, emulating the way humans reason through a problem. Rather than directly outputting the final answer, LLMs are required to generate a step-by-step reasoning process, which can help models handle more complex tasks. Self-consistency with CoT (CoT-SC) (Wang et al., 2023a) improves upon CoT by generating different thought processes for the same problem and the output decision can be more reliable by exploring a richer set of thoughts. In this paper, we utilize the CoT-SC paradigm to collaborate with various agents in EDAid to perform EDA task planning and script generation for EDA flow automation.

In-Context Learning. In-context learning (Min et al., 2022) has emerged as a transformative paradigm in machine learning, characterized by the meticulous training of models to perform specific tasks through examples and directives provided within an interactive, conversational framework. The ability of in-context learning emerges (Wei et al., 2022a) in large-scale, versatile LLMs (Achiam et al., 2023; Anthropic, 2024; Dubey et al., 2024). These models demonstrate an impressive ability to leverage their broad knowledge across various downstream tasks through in-context learning (Brown et al., 2020). In our research, we apply few-shot prompts to enhance the performance and reliability of each agent in EDAid based on in-context learning.

LLM-powered Agent System. Single-agent systems driven by LLMs have demonstrated remarkable cognitive capabilities (Sumers et al., 2023; Wang et al., 2024; Xi et al., 2023; Wu et al., 2024). These LLM-powered agents can decompose complex tasks into manageable subtasks (Khot et al., 2022) and methodically think through each component to make better decisions. Moreover, the tool-

EDA Task

```
Can you help me check whether the provided script can complete the task and
provide the reason.
Here is the task:
I want to optimize performance at final stage for my design asjdk on nangate45.
Here is the script:
def tune_performance():
    params = {
        "tns_end_percent": {"minmax": [0, 100], "step": 5}
    }
    def optimize(tns_end_percent):
        eda = chateda()
        eda.setup(design_name="asjdk", platform="nangate45")
        eda.run_synthesis()
        eda.floorplan()
        eda.placement()
        eda.cts(tns_end_percent=tns_end_percent)
        return eda.get_metric(stage="cts", metrics=["performance"])
    best_params = tune(optimize, params)
tune_performance()
If the script can't complete the task, please provide a script that can complete
the task.
```

Task Planning Pathway

```
The task is to optimize the performance of the "asjdk" design on the "nangate45"
platform. The provided script attempts to tune the "tns_end_percent" parameter
during the clock tree synthesis (CTS) stage to optimize the performance. However,
the script only considers the performance after CTS and not at the final stage.
```

EDA Script Generation

```
def tune_performance():
    params = {
        "tns_end_percent": {"minmax": [0, 100], "step": 5}
    }
    def optimize(tns_end_percent):
        eda = chateda()
        eda.setup(design_name="asjdk", platform="nangate45")
        eda.run_synthesis()
        eda.floorplan()
        eda.placement()
        eda.cts(tns_end_percent=tns_end_percent)
        eda.global_route()
        eda.detail_route()
        eda.density_fill()
        eda.final_report()
        return eda.get_metric(stage="final", metrics=["performance"])
    best_params = tune(optimize, params)
tune_performance()
```

EDA Task

```
Can you help me check whether the provided script can complete the task and
provide the reason.
Here is the task:
I want to perform cts for my design asjdk on nangate45.
Here is the script:
def perform_cts(tns_end_percent):
    eda = chateda()
    # Setup EDA tool
    eda.setup(design_name="asjdk", platform="nangate45")
    # Run logic synthesis
    eda.run_synthesis()
    eda.floorplan()
    # Run placement
    eda.placement(tns_end_percent=tns_end_percent)
    # Run CTS
    eda.cts()
perform_cts(10)
If the script can't complete the task, please provide a script that can complete
the task.
```

Task Planning Pathway

```
The task is to perform Clock Tree Synthesis (CTS) for the design named "asjdk" on
the "nangate45" platform. The provided script attempts to do this, but it has an
error in the placement step. The "placement" method does not have a parameter
called "tns_end_percent". The "tns_end_percent" is actually a parameter for the
"cts" method.
Here's the correct way to complete the task:
1. Setup the EDA tool with the design name and platform.
2. Run logic synthesis.
3. Perform floorplanning.
4. Run placement.
5. Run CTS with the desired "tns_end_percent".
```

EDA Script Generation

```
def perform_cts(tns_end_percent):
    eda = chateda()
    # Setup EDA tool
    eda.setup(design_name="asjdk", platform="nangate45")
    # Run logic synthesis
    eda.run_synthesis()
    # Run floorplan
    eda.floorplan()
    # Run placement
    eda.placement()
    # Run CTS
    eda.cts(tns_end_percent=tns_end_percent)
perform_cts(10)
```

Figure 7: Error correctness. The decision-making agent (role R_1) can identify and fix errors in given EDA scripts accurately according to the given EDA tasks.

calling capability (Qin et al., 2023) of LLMs enables agent systems to leverage external resources and tools, allowing them to operate more effectively in various scenarios. Developed based on the single-agent system, several studies (Hong et al., 2024; Wang et al., 2023b; Du et al., 2023; Hao et al., 2023) have enhanced the problem-solving abilities of LLMs by integrating discussions among multiple agents. The collaboration of multiple autonomous agents, each equipped with unique strategies, can address more dynamic and complex tasks. In this work, multiple agents in EDAid collaborate with divergent thoughts to automate EDA flow via complex long-chain EDA tool-calling.

7 Conclusion

Automating EDA flow by interfacing EDA tools via APIs is imperative to enhance efficiency in electronic design processes. In this study, we first introduce the ChipLlama-powered agent, collaborating with few-shot prompts for EDA flow automation. Specifically, ChipLlama models are expert LLMs instruction fine-tuned for the EDA flow automation, which achieve the SOTA performance in EDA script generation and demonstrate versatility across different platforms. Meanwhile, different few-shot CoT prompts can guide LLMs to generate different task planning pathways for divergent thoughts generation. Building with divergent thoughts generation and decision-making, we present EDAid, a novel multi-agent collaboration system that utilizes multiple agents for EDA tasks. This system adeptly handles intricate EDA tasks assigned by designers, thereby automating the EDA workflow effectively. Our experiments demonstrate the effectiveness of our EDAid. Moreover, extensive experiments show the significant performance of our ChipLlama models compared to other LLMs in automating the EDA flow. In conclusion, we anticipate that our work can catalyze the evolution of next-generation EDA tools, inspiring advancements in the field.

Limitation

Our multi-agent collaboration system, EDAid, introduces inference latency compared to the single-agent system, considering that the divergent thoughts and decision-making process require multiple inference steps of LLMs. We will provide more discussion in Appendix B.

Acknowledgement

This work is partially supported by The Research Grants Council of Hong Kong SAR (No. RFS2425-4S02, No. CUHK14211824, No. CUHK14210723), and AI Chip Center for Emerging Smart Systems (ACCESS), Hong Kong SAR.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. GPT-4 Technical Report. *arXiv preprint arXiv:2303.08774*.

Tutu Ajayi and David Blaauw. 2019. OpenROAD: Toward a self-driving, open-source digital layout implementation tool chain. In *Government Microcircuit Applications & Critical Technology Conference (GOMACTech)*.

Anthropic. 2024. Claude. https://www.anthropic.com/index/claude-3.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. In *Annual Conference on Neural Information Processing Systems (NIPS)*, volume 33, pages 1877–1901.

Pinhong Chen, Desmond A Kirkpatrick, and Kurt Keutzer. 2001. Scripting for EDA tools: a case study. In *IEEE International Symposium on Quality Electronic Design (ISQED)*.

Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 2021. 8-bit optimizers via block-wise quantization. In *International Conference on Learning Representations (ICLR)*.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2024. QLoRA: Efficient Finetuning of Quantized LLMs. In *Annual Conference on Neural Information Processing Systems (NIPS)*, volume 36.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B Tenenbaum, and Igor Mordatch. 2023. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783*.

Rui Hao, Linmei Hu, Weijian Qi, Qingliu Wu, Yirui Zhang, and Liqiang Nie. 2023. ChatLLM network: More brains, more intelligence. *arXiv preprint arXiv:2304.12998*.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework. In *International Conference on Learning Representations (ICLR)*.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2022. Decomposed prompting: A modular approach for solving complex tasks. *arXiv preprint arXiv:2210.02406*.

Xingquan Li, Zengrong Huang, Simin Tao, Zhipeng Huang, Chunan Zhuang, Hao Wang, Yifan Li, Yihang Qiu, Guojie Luo, Huawei Li, et al. 2024. iEDA: An Open-source infrastructure of EDA. In *IEEE/ACM Asia and South Pacific Design Automation Conference (ASPDAC)*, pages 77–82.

Mingjie Liu, Teodor-Dumitru Ene, Robert Kirby, Chris Cheng, Nathaniel Pinckney, Rongjian Liang, Jonah Alben, Himyanshu Anand, Sanmitra Banerjee, Ismet Bayraktaroglu, et al. 2023. ChipNeMo: Domain-Adapted LLMs for Chip Design. *arXiv preprint arXiv:2311.00176*.

Sewon Min, Mike Lewis, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2022. Metaicl: Learning to learn in context. In *Annual Meeting of the Association for Computational Linguistics (ACL)*.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, et al. 2023. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. *arXiv preprint arXiv:2307.16789*.

Theodore R Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L Griffiths. 2023. Cognitive architectures for language agents. *arXiv preprint arXiv:2309.02427*.

Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, et al. 2024. A survey on large language model based autonomous agents. *Frontiers of Computer Science*, 18(6):186345.

Xuezhi Wang et al. 2023a. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations (ICLR)*.

Zhenhailong Wang, Shaoguang Mao, Wenshan Wu, Tao Ge, Furu Wei, and Heng Ji. 2023b. Unleashing the emergent cognitive synergy in large language models: A task-solving agent through multi-persona self-collaboration. *arXiv preprint arXiv:2307.05300*.

Jason Wei, Yi Tay, Rishi Bommasani, Colin Raffel, Barret Zoph, Sebastian Borgeaud, Dani Yogatama, Maarten Bosma, Denny Zhou, Donald Metzler, et al. 2022a. Emergent Abilities of Large Language Models. *Transactions on Machine Learning Research (TMLR)*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022b. Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. In *Annual Conference on Neural Information Processing Systems (NIPS)*, volume 35, pages 24824–24837.

Yuxiang Wei, Zhe Wang, Jiawei Liu, Yifeng Ding, and Lingming Zhang. 2023. Magicoder: Source code is all you need. *arXiv preprint arXiv:2312.02120*.

Haoyuan Wu, Zhuolun He, Xinyun Zhang, Xufeng Yao, Su Zheng, Haisheng Zheng, and Bei Yu. 2024. ChatEDA: A Large Language Model Powered Autonomous Agent for EDA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*.

Zhiheng Xi, Wenxiang Chen, Xin Guo, Wei He, Yiwen Ding, Boyang Hong, Ming Zhang, Junzhe Wang, Senjie Jin, Enyu Zhou, et al. 2023. The rise and potential of large language model based agents: A survey. *arXiv preprint arXiv:2309.07864*.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2023. MetaMath: Bootstrap Your Own Mathematical Questions for Large Language Models. *arXiv preprint arXiv:2309.12284*.

A Details of Hybrid Instruction Tuning

We show the proportion of MathInstruct (Yu et al., 2023), CodeInstruct (Wei et al., 2023), and EDAInstruct (Wu et al., 2024) datasets for hybrid instruction tuning of ChipLlama models in Table 5.

Firstly, the MathInstruct dataset focuses on CoT reasoning, strengthening the logical reasoning skills required for EDA task planning. Then, the CodeInstruct dataset presents complex coding tasks and solutions, offering coding skills for EDA script generation. Finally, we utilize the EDAInstruct dataset which provides the EDA domain-specific knowledge and instructions for EDA flow automation. The composition of these datasets creates a hybrid corpus meticulously designed to expand the capabilities of ChipLlama models.

	MathInstruct	CodeInstruct	EDAInstruct
Proportion	80K	100K	8K

Table 5: The proportion of MathInstruct, CodeInstruct and EDAInstruct datasets for hybrid instruction tuning.

B Agents in EDAid

In our experiments, we use three agents to generate divergent thoughts and one agent to compute the probabilities of these thoughts to make the final decision. When the number of agents generating divergent thoughts is less than three, the stability and accuracy of EDA flow automation with EDAid improve with the addition of more agents. However, once the number of agents reaches three, the performance tends to saturate, meaning that further increases in the number of agents do not necessarily lead to significant improvements in performance. In practical applications, considering the costs associated with real-world EDA flows, as we mentioned before, an appropriate increase in the number of agents can still be acceptable if it enhances the system's overall stability and reliability.

C More Case Studies

We provide more case studies to figure out how our EDAid resolves the given EDA task. As illustrated in Figure 8, we provide two EDA tasks and their corresponding task planning pathways and generated EDA scripts. Both EDA tasks require the system to provide a parameter-tuning solution. Our system appropriately grasps the need for the given EDA task and shows an excellent understanding of the details of each API interface parameter.

EDA Task

```
My "h264_encoder" design needs to meet tough area constraints in "gf180". Please
suggest how I can automatically optimize cts and floorplan parameters to minimize
area.
```

Task Planning Pathway

```
The task is to minimize the area of the "h264_encoder" on the "gf180" platform by
optimizing the Clock Tree Synthesis (CTS) and floorplanning parameters.
To complete this task, we will follow these steps:
1. Initialize the EDA tool using "setup()" with the design name and platform.
2. Define a function that takes the floorplan aspect ratio, core utilization, core
margins, macro place halo, macro place channel, and tns_end_percent as input
parameters.
Inside the function, call the appropriate methods of the chateda instance in the
correct order, passing the parameter values and finally return the area metric.
    2a. Call "run_synthesis()" with default parameters.
    2b. Use "floorplan()" with aspect ratio, core utilization, core margins, macro
place halo, and macro place channel.
    2c. Execute "placement()" with default parameters.
    2d. Invoke "cts()" with the percentage of timing violations to fix.
    2e. Complete routing and perform finishing steps.
    2f. Obtain the final area metric using "get_metric()" after the entire
process.
3. Define the parameter ranges.
4. Utilize tune function to perfrom DSE and find the best combination.
```

EDA Script Generation

```python
def tuning_func(aspect_ratio, core_utilization, core_margins,
                macro_place_halo, macro_place_channel, tns_end_percent):
    tool = chateda()
    tool.setup("h264_encoder", "gf180")
    tool.run_synthesis()
    tool.floorplan(
        core_aspect_ratio=aspect_ratio,
        core_utilization=core_utilization, core_margins=core_margins,
        macro_place_halo=macro_place_halo, macro_place_channel=macro_place_channel
    )
    tool.placement()
    tool.cts(tns_end_percent=tns_end_percent)
    tool.global_route()
    tool.detail_route()
    tool.density_fill()
    tool.final_report()
    return tool.get_metric("final", ["area"])
params = {
    "aspect_ratio": {"minmax": [0.5, 2], "step": 0.25},
    "core_utilization": {"minmax": [50, 90], "step": 10},
    "core_margins": {"minmax": [6, 12], "step": 2},
    "macro_place_halo": {"minmax": [2, 6], "step": 1},
    "macro_place_channel": {"minmax": [2, 6], "step": 1},
    "tns_end_percent": {"minmax": [50, 90], "step": 10}
}
tune(tuning_func, params)
```

EDA Task

```
The project's name is "hello", and we are using the "asap7" platform. We want to
perform an optimization of ppa at the final stage. Try to find the best
combinations of floorplan parameters.
```

Task Planning Pathway

```
To complete this task, we will follow these steps:
1. Initialize an object of the "chateda" class. This object will be used to
invoke the various methods provided in the APIs.
2. Define a function that takes the floorplan parameters as input and returns the
PPA metrics at the final stage. This function will be passed to the "tune" method
for optimization.
3. Inside the function, perform the following steps:
    3a. Setup the EDA tool for the "hello" project using the "asap7" platform.
    3b. Run logic synthesis.
    3c. Run floorplan with the given parameters.
    3d. Run placement.
    3e. Run clock tree synthesis (CTS).
    3f. Run global routing.
    3g. Run detail routing.
    3h. Run density fill.
    3i. Run final report.
    3j. Return the PPA metrics at the final stage.
4. Define the parameter ranges for the floorplan parameters.
5. Pass the function and parameter ranges to the "tune" method for optimization.
```

EDA Script Generation

```python
def tuning_func(core_utilization, core_aspect_ratio, core_margins,
                macro_place_halo, macro_place_channel):
    tool = chateda()
    tool.setup("hello", "asap7")
    tool.run_synthesis()
    tool.floorplan(
        core_utilization=core_utilization,
        core_aspect_ratio=core_aspect_ratio,
        core_margins=core_margins,
        macro_place_halo=macro_place_halo,
        macro_place_channel=macro_place_channel)
    tool.placement()
    tool.cts()
    tool.global_route()
    tool.detail_route()
    tool.density_fill()
    tool.final_report()
    return tool.get_metric("final", ["power", "performance", "area"])
params = {
    "core_utilization": {"minmax": [50, 90], "step": 10},
    "core_aspect_ratio": {"minmax": [0.5, 2], "step": 0.25},
    "core_margins": {"minmax": [1, 5], "step": 1},
    "macro_place_halo": {"minmax": [1, 5], "step": 1},
    "macro_place_channel": {"minmax": [1, 5], "step": 1}
}
tune(tuning_func, params)
```

Figure 8: Case Studies of EDA flow automation with our EDAid powered by ChipLlama models. Each case provides an EDA task, its corresponding task planning pathway and the generated EDA script.