SpecLLM: Exploring Generation and Review of VLSI Design Specification with Large Language Model

Mengming Li, Wenji Fang, Qijun Zhang and Zhiyao Xie*

Hong Kong University of Science and Technology
Hong Kong, China
*Coresponding Author: eezhiyao@ust.hk

2025 International Symposium of Electronics Design Automation (ISEDA) | 979-8-3315-3696-1/25/$31.00 ©2025 IEEE | DOI: 10.1109/ISEDA65950.2025.11100410

***Abstract*—The development of architecture specifications is an initial and fundamental stage of the integrated circuit (IC) design process. Traditionally, architecture specifications are crafted by experienced chip architects, a process that is not only time-consuming but also error-prone. Mistakes in these specifications may significantly affect subsequent stages of chip design. Despite the presence of advanced electronic design automation (EDA) tools, effective solutions to these specification-related challenges remain scarce. Since writing architecture specifications is naturally a natural language processing (NLP) task, this paper pioneers the automation of architecture specification development with the advanced capabilities of large language models (LLMs).**

We propose a structured definition of architecture specifications, categorizing them into three distinct abstraction levels. Based on this definition, we create and release a specification dataset[1] by methodically gathering 46 architecture specification documents from various public sources. Leveraging our definition and dataset, we explore the application of LLMs in two key aspects of architecture specification development: (1) Generating architecture specifications, which includes both writing specifications from scratch and converting RTL code into detailed specifications. (2) Reviewing existing architecture specifications. We got promising results indicating that LLMs may revolutionize how these critical specification documents are developed in IC design nowadays.

***Index Terms*—Architecture specification, LLM, dataset.**

I. Introduction

Developing architecture specifications is a critical initial step in the process of IC design. It lays the foundational framework and guidelines necessary for the subsequent stages of design and development. Traditionally, the task of writing and reviewing architecture specifications is undertaken by skilled chip architects. This process, while expertise-driven, tends to be time-consuming and can be susceptible to human errors, soliciting a more automated methodology to enhance efficiency and accuracy.

In recent years, LLMs such as ChatGPT [1] have showcased remarkable capabilities in the field of artificial intelligence, with a wide range of applications from question answering to content creation. The growth of chip computing power will endow LLMs with greater capabilities. Consequently, researchers have started to investigate the potential of LLMs in augmenting the process of chip design, reversely enhancing the computing power of the chips themselves. For instance, recent studies [2]–[9] have utilized LLMs to generate RTL code like Verilog. Other works develop LLM-based solution to control EDA tools [9], [10], design AI accelerator architectures [11], [12], hardware security assertion generation [13], fix security bugs [14], etc. These research efforts imply a promising future for LLMs in chip design. In this paper, we conduct a pioneering investigation in the potential and practicality of LLMs in processing specifications. To the best of our knowledge, there has been no prior design automation or LLM research focusing on this important topic.

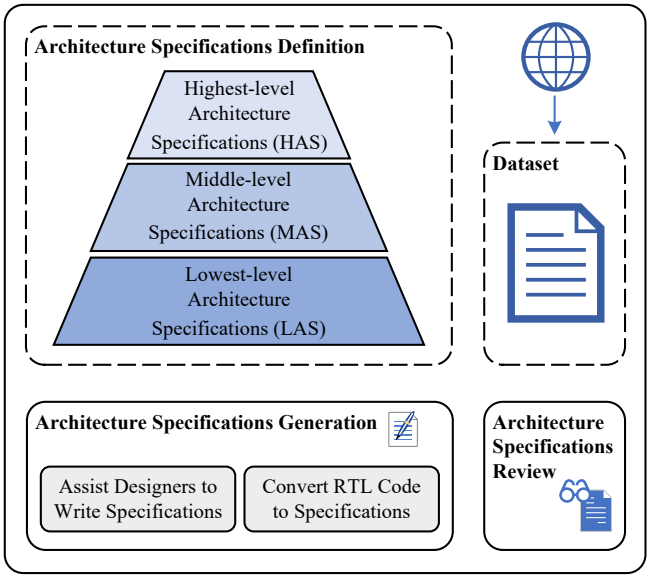

Fig. 1: The overall structure of this paper. We first propose basic definitions and an organized dataset dedicated to architecture specifications. Leveraging them, we explore the use of LLMs in the generation and review of architecture specifications.

This paper focuses on employing LLMs to address the challenges inherent in the traditional management of architecture specifications. Figure 1 highlights the overall structure of this paper. Considering the absence of a formal definition or categorization of architecture specifications, we start with investigating existing architecture specifications across a diverse range of products. Then we categorize architecture specifications into three levels: Highest-level Architecture Specification (HAS), Middle-level Architecture Specification (MAS), and

This work is supported by Hong Kong Research Grants Council (RGC) CRF Grant C6003-24Y and ACCESS – AI Chip Center for Emerging Smart Systems, sponsored by InnoHK, Hong Kong SAR. We thank HKUST Fok Ying Tung Research Institute and National Supercomputing Center in Guangzhou Nansha Sub-center for computational resources.

[1]https://github.com/hkust-zhiyao/SpecLLM

Lowest-level Architecture Specification (LAS).

Building upon our definition, we have assembled a comprehensive dataset that includes 46 public architecture specifications from various types of products. Leveraging this dataset, we investigate the application of LLMs in both generation and reviewing architecture specifications. Regarding specification generation, we suggest two potential approaches. The first is to simplify the process of writing architecture specifications for designers, making it more efficient and less error-prone. The second approach is applicable when architecture specifications are absent for an already implemented chip. In such scenario, we can transform the RTL code back into architecture specifications, essentially reconstructing the original design documentation from the implemented code. According to our experiment, when generating specifications for simple logic circuits, the majority of the human tasks could be done by LLMs. We are optimistic that, even for more complex logic circuits, it is promising for LLMs to progressively take over a large portion of the work.

We also demonstrate that the LLMs can efficiently review the architecture specifications. We first propose our definition of various types of defects in architecture specification document. Building on this, we utilize these defects as the target responses and develop specific prompts, seeking for LLM's review. As for the publicly available architecture specifications, our experiments have demonstrated that the LLMs could provide valuable feedback for enhancing these documents. Moving forward, we extend our research to include the evaluation of the review results generated by the LLMs, streamlining the review process and enhancing the reliability of the outcomes.

Our key contributions are summarized below:

- We provide structured definitions of architecture specifications, facilitating efficient utilization of LLMs in developing architecture specifications. (Section II)
- We generate a dataset of design specifications by methodically collecting and systematically organizing architecture specification documents from a variety of online sources. (Section III)
- We explore the use of LLMs as tools for generating architecture specifications, including assisting designers in writing these specifications and converting RTL code into comprehensive specifications. Our findings suggest that LLMs hold considerable promise in efficiently generating architecture specifications. (Section IV)
- We explore the use of LLMs in the review of architecture specifications. We identify various potential defects that may arise in these specifications. Based on these identified defects, we have crafted specific processes and prompts to guide the LLMs in their review. We utilize publicly available architecture specifications to evaluate our methodologies. Our experimental results demonstrate that LLMs are capable of providing valuable feedback for improving these documents. (Section V)

II. Our Definition on Architecture Specification

We define the architecture specification as the document that describes the chip architecture prior to RTL coding. Writing architecture specifications for the target chip is usually the starting point of the IC design flow. The term **architecture specification** is specifically chosen to emphasize that the document captures the architectural aspects of a chip, distinguishing it from general specifications. We categorize the architecture specifications into three levels.

- **The Highest-level Architecture Specification (HAS)** establishes standards applicable to a range of products. A notable example is the RISC-V specifications [16]. It defines the ISA specifications (e.g., instruction formats, register usages) and Non-ISA specifications (e.g., trace). Designing specific RISC-V chips should comply with these specifications.
- **The Middle-level Architecture Specification (MAS)** outlines the high-level architecture of a single product. These specifications encompass the essential information required to profile the chip design. For example, one MAS for a RISC-V CPU may include an overview of the microarchitectures (e.g., block diagram) and their primary parameters (e.g., cache size).
- **The Lowest-level Architecture Specification (LAS)** details the microarchitecture design of a single product. Unlike HAS and MAS, LAS should give the implementation details for each microarchitecture, which may involve the ports, internal signals, pipelines, and the associated descriptions. By reading them, the designers are expected to write the corresponding RTL code correctly.

III. Dataset Overview

Our dataset [15] includes a variety of public architecture specifications, organized by product types like CPU, SoC, Accelerator, Bus and Network, Arithmetic and Crypto, and by levels such as HAS, MAS, and LAS. Our collection has yielded several notable findings.

Availability. For the HAS, MAS, or LAS, the architecture specifications related to RISC-V are among the most accessible and widely available. For the Arm and X86 ecosystem, the availability of formal architecture specifications is weaker.

Standard. The available open specification documents exhibit a variety of writing styles and lack unified writing standards. This variation in writing styles and lack of standardized formats in architecture specifications pose additional challenges for LLMs in handling and accurately interpreting these documents.

Length. The length of HAS spans from just over 100 pages to more than 1000 pages. The lengths of MAS and LAS can vary widely, with their extent directly correlating to the complexity of the chip designs they describe.

IV. LLM Generates Architecture Specification

A. Motivation

LLMs are promising in assisting designers in writing architecture specifications. We mainly focus on using LLMs to

TABLE I: Our proposed dataset for architecture specifications, including approximately 46 specification documents. All of these documents are available in our open-sourced repository [15].

Type	**Highest-level Architecture Specifications (HAS)**	**Middle-level Architecture Specifications (MAS)**	**Lowest-level Architecture Specifications (LAS)**
CPU	• RISC-V ISA Specifications, Unprivileged Specification • RISC-V ISA Specifications, Privileged Specification • ARMv8-M Architecture Reference Manual • Intel 64 and IA-32 Architectures Software Developer's Manual • The SPARC Architecture Manual, Version 9 • OpenRISC 1000 Architecture Manual	• *The NEORV32 RISC-V Processor: Datasheet • *OpenSPARC T1 Microarchitecture Specification • *OpenSPARC T2 Core Microarchitecture Specification • E31 Core Complex Manual • E51 Core Complex Manual • Arm Cortex-A78 Core Technical Reference Manual • Arm Cortex-X2 Core Technical Reference Manual • Arm Neoverse-N2 Core Technical Reference Manual • Intel 64 and IA-32 Architectures Optimization Reference • OpenRISC 1200 IP Core Specification	• The NEORV32 RISC-V Processor: Datasheet • OpenSPARC T1 Microarchitecture Specification • OpenSPARC T2 Core Microarchitecture Specification • Amber 2 Core Specification • LXP32, a lightweight open source 32-bit CPU core, Technical Reference Manual • OpenMSP430, Texas Instruments • NEO430, based on the Texas Instruments MSP430(TM) ISA
SoC	• Efficient Trace for RISC-V • RISC-V External Debug Support • RISC-V IOMMU Architecture Specification • RISC-V Advanced Interrupt Architecture • RISC-V Platform-Level Interrupt Controller Specification	• Freedom E310-G000 Manual • Freedom U540-C000 Manual	• #The NEORV32 RISC-V Processor: Datasheet • OpenSPARC T2 System-On-Chip (SoC) Microarchitecture Specification
Accelerator	• RISC-V "V" Vector Extension • Intel Advanced Performance Extensions (Intel APX) Architecture Specification • Intel Advanced Vector Extensions 10 (Intel AVX10) Architecture Specification		• NVIDIA Deep Learning Accelerator (NVDLA), Hardware Architectural Specification
Bus & Network	• TileLink Specification • AMBA5 CHI Architecture Specification • AMBA5 ACE Protocol Specification (superseded by CHI) • AMBA5 AXI Protocol Specification • AMBA4 AXI and ACE Protocol Specification		• 10GE MAC Core Specification • Ethernet IP Core Specification • I2C-Master Core Specification • UART to Bus Core Specification
Arithmetic			• Elliptic Curve Group Core Specification • Tate Bilinear Pairing Core Specification • Tiny Tate Bilinear Pairing Core Specification
Crypto			• AES Core Specification • SHA3 Core Specification

⋆Notes: *NEORV32, OpenSPARC T1, OpenSPARC T2 include chapters meeting the criterion of MAS. #NEORV32 encompasses chapters describing the SoC implementation.

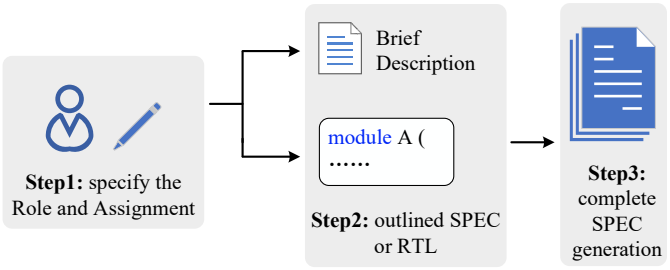

Fig. 2: LLM-based applications on architecture specification generation.

Prompt:
From now on, you act as a chip architect writing the architecture specification. This specification must encompass detailed descriptions of microarchitectures, including aspects such as ports, internal signals, state machines. Subsequently, I will assign you a specific module to focus on in your writing.

Fig. 3: An example prompt to initiate generating LAS. This generation is based on the designers' brief description.

automatically generate the MAS and LAS. We explore two LLM-based applications:

1) LLMs can reduce human effort by writing architecture specifications (Section IV-B).
2) Many open-source chip designs lack comprehensive and formal architecture specifications. LLMs could reversely convert the RTL code into detailed specification documents. This reverse engineering process can bridge the gap in the documentation and enhance understanding and accessibility of the chip designs (Section IV-C).

In the upcoming sections, we will leverage the widely-used commercial LLM product—GPT [1], to demonstrate how this technology can be effectively exploited to enhance the process of writing architecture specifications.

B. Assist Designers to Write Architecture Specifications

Figure 3 presents an example of a prompt to initiate the generation of LAS. This example, referencing [17], indicates the specific role expected to be performed by the LLMs, and the expected response format. Then we employ the prompt in Figure 4 to direct GPT generating the architecture specifications. This particular example, sourced from [4], has been selected due to its moderate level of complexity, making it suitable for testing purposes. Figure 5 presents the outcomes of GPT. The experimental results demonstrate that the GPT can accurately understand the questions. These specifications, spanning from the high-level functional description to the low-level state machines, offer in-depth information for subsequent RTL coding. Yet, the design of a perpetual calendar is a classical task in the field of circuit design. The LLMs have abundant corpus about it. The commercial products, usually include internal IPs, which are not used to train the external LLMs like ChatGPT. We believe it is non-trivial to generate that kind of architecture specifications. However, our experiments at least demonstrate that the LLMs possess the potential to conduct the task of generating architecture specifications. The speed of the

Prompt:
Please write an architecture specification regarding a perpetual calendar. The starting value of Secs, Mins, and Hours are all 0. Both Secs and Mins loop continuously from 0 to 59. When Secs=59, Min increases by 1 at the next cycle, and when Min=59 && Secs=59, Hours increases by 1 at the next cycle. Hours is counted from the 0-23 cycle.

Fig. 4: An example prompt to request GPT generating an architecture specification for perpetual calendar. The response is shown in Figure 5.

Response:
Architecture Specification for a Perpetual Calendar Module

Overview
This document outlines the architectural specification for a Perpetual Calendar Module (PCM). The PCM is responsible for .....

Functional Description
Time Units
Seconds (Secs): This unit ranges from 0 to 59. It resets to 0 after reaching 59.
Minutes (Mins): This unit also ranges from 0 to 59. It increments by 1 when Secs roll over from 59 to 0.
.....

Microarchitecture
Components
Counter Units: Three independent counters for Secs, Mins, and Hours.
Control Logic: Determines the rollover conditions and triggers the increment of Mins and Hours.
.....

Interface
Input: Clock signal.
Output: Current time in hours, minutes, and seconds.

State Machines
Seconds Counter (Secs)
State 0-58: Increment Secs by 1 each clock cycle.
State 59: Reset Secs to 0 on the next clock cycle and trigger a Min increment signal.
Minutes Counter (Mins)
.....

Fig. 5: A GPT response example to the prompt in Figure 4.

LLMs in performing this task, compared to the experienced chip architects, still has significant advantages.

C. Convert RTL code to Architecture Specifications

Figure 6 showcases the prompt employed for converting the RTL code into LAS. Following this prompt, we provide the RTL code to the LLM, which can be in the form of textual format or as a source code file. In our example, we adopt the Verilog file that is open-sourced by RTLLM [4] and it is uploaded to our GitHub repo. In summary, the generated architecture specifications preserve the correct meaning. We conduct experiments to analyze this phenomenon. Our findings indicate that the provided RTL code confines the scope for the specification generation. Without the RTL code, the LLM may generate the architecture specifications on the basis of the previously trained corpus, which exhibits diverse explanations for the single circuit design. On the contrary, after we offer

Prompt:
From now on, you act as a chip architect writing the architecture specification. This specification must encompass detailed descriptions of microarchitectures, including aspects such as ports, internal signals, state machines. Subsequently, I will provide you with the RTL code. Please read the RTL code and generate the corresponding architecture specification.

Fig. 6: An example prompt to initiate writing architecture specifications on the basis of RTL code.

the LLM with a unique RTL code, its responses are likely to be more orientated.

V. LLM Reviews Architecture Specification

A. Motivation

The precision of architecture specifications directly affect the overall quality of chip designs. Ensuring these specifications are correct is crucial for the successful development and functionality of the chips. Traditionally, the chip company should spend efforts to review architecture specifications. Analogous to the reasons mentioned in Section IV-A, reviewing architecture specifications also requires significant human efforts. LLMs can also help reviewers in these tasks. At least, the reviewers could use these tools to get an overview of the specification files and obtain some comments from them. Fortunately, as we will demonstrate in Section V-B and Section V-C, the LLMs show promise in achieving more profound objectives in the review of architectural specifications, going beyond basic overviews to provide detailed insights.

B. Defects Category

TABLE II: Potential Defects Occurred in Different Levels Architecture Specifications

Type	Potential Defects
Common Case	• Typographical Error • Inconsistence or Contradiction Error • Incomplete or Unclear Error
Level Specific	• Combinational Loops Error (LAS) • Uninitialized Register Value Error (LAS) • Improvement for Micro-architectural Design (LAS) • Improvement for Architectural Design (MAS)
Level Spanning	• Inconsistence or Contradiction Error (Across Various Levels)

Initially, we have pinpointed various types of potential defects that may occur in architecture specifications. Table II summarizes these defects, categorizing them according to the level of architecture specifications.

In many instances, there are defects that could manifest in all three tiers of architecture specifications: HAS, MAS, and LAS. They include the *Typographical Error*, *Inconsistence or Contradiction Error*, and *Incomplete or Unclear Error*. The *Inconsistence or Contradiction Error* denotes situations within a single specification file where either two concepts describing the same object are inconsistent, or two related concepts are contradictory. The *Incomplete or Unclear Error* refers to instances where certain concepts lack essential information, resulting in sentences that are open to ambiguous interpretations.

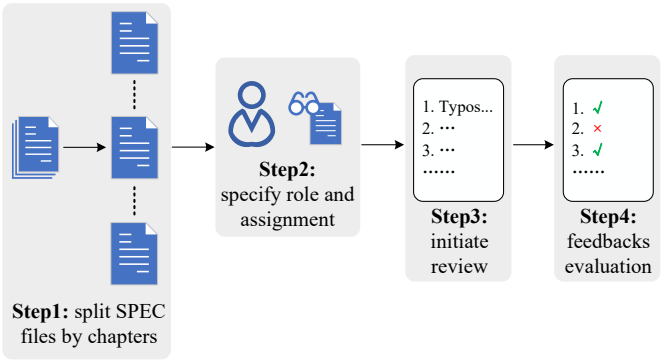

Fig. 7: The flow of LLM-based architecture specification review.

Furthermore, the potential defects could also be level-specific, indicating that HAS, MAS, or LAS may each have distinct defects. For example, LAS might include details about port connections between various modules. LLMs can be employed to scrutinize the LAS for identifying potential *Combinational Loops Error*. Such scrutiny, however, may not be applicable to HAS and MAS. In addition to locating the writing issues, we also identify some high-level objectives for the review of architecture specification, such as the *Improvement for Architectural Design* and *Improvement for Micro-architectural Design*. These objectives are aimed at leveraging the potential of the review process to enhance both the architectural and micro-architectural design aspects. This goes beyond mere error correction, focusing on the overall optimization and refinement of the design.

Additionally, these defects might span across different levels. One example of this is the *Inconsistence or Contradiction Error (Across Various Levels)*, that the lower-level architecture specifications should not contradict the contents established in the higher-level specifications. Note that it is not equivalent to the *Inconsistence or Contradiction Error* established for the common case. The former indicates the inconsistence or contradiction between different levels of architecture specification files, while the latter confines the error in one individual specification file.

C. Architecture Specification Review

We exploit the potential defects in architecture specifications as the types of feedback (Section V-B) we aim to receive from the review conducted by the LLMs. As shown in Figure 7, we have developed a methodology to guide the LLMs in reviewing architecture specifications. This method entails: (1) splitting the specification file into individual chapters, (2) specifying the role and assignment of the LLM, and (3) asking the LLMs to conduct a review immediately upon the upload of each chapter. We observe that splitting the specification file into chapters and uploading each one separately for review, can provide clear boundaries for the LLMs to analyze and respond to each

Prompt:
From now on, you act as a chip architect reviewing the architecture specification. For the following document, please find any typo, inconsistence/contradiction, incomplete/unclear and propose the corresponding improvement suggestions. The responding format should separately list the findings about typo, inconsistence/contradiction, incomplete/unclear and the corresponding improvement suggestions.

Fig. 8: An example prompt to initiate architecture specification review.

Prompt:
Please review the chapter 1 first, and give me specific and comprehensive reviewing results. The responding format should comply with the previous requirements.
(Note: attach the divided file that represents chapter 1 of the architecture specification file)

Fig. 9: An example prompt to review the specification document. The response is shown in Figure 10.

Response:
Chapter 1 Review of the Amber Core Specification

Typos:
1. "seperate" (in the description of Amber 25) should be "separate."
2. "Mhz" (in the capability description of Amber 25) should be "MHz."
.....

Inconsistencies/Contradictions:
1. The document states that the Amber 23 and Amber 25 cores are 100% software compatible, yet they have differing pipeline stages and cache architectures. This could potentially lead to different performance characteristics not addressed in the compatibility claim.

Incomplete/Unclear Sections:
1. The description of the Wishbone interface is vague. More detailed specifications or references to the interface's documentation would improve clarity.
.....

Improvement Suggestions:
1. **Typos**:.....
2. **Inconsistencies/Contradictions:**.....
3. **Incomplete/Unclear Sections:**.....
.....

Fig. 10: A GPT response example to the prompt in Figure 9.

chapter individually. In the subsequent sections, we will use part of the architecture specifications, as shown in Table I, to demonstrate LLM's strengths and weaknesses in reviewing architecture specifications.

We begin by using prompts, as shown in Figure 8, to initiate the review of architecture specifications. Next, we upload the specification file selected for testing and provide the prompt, as illustrated in Figure 9, directing GPT to review each chapter. In our strategy, each review task for different chapters is paired with its corresponding chapter file, ensuring that each chapter is reviewed in isolation and in context. Figure 10 displays the in-depth review results specifically for chapter 1 of the Amber Core Specification. In general, we have observed that GPT is adept at identifying typographical errors. Furthermore, in more complex cases such as inconsistence or contradiction errors, as well as incomplete or unclear errors, GPT is also capable

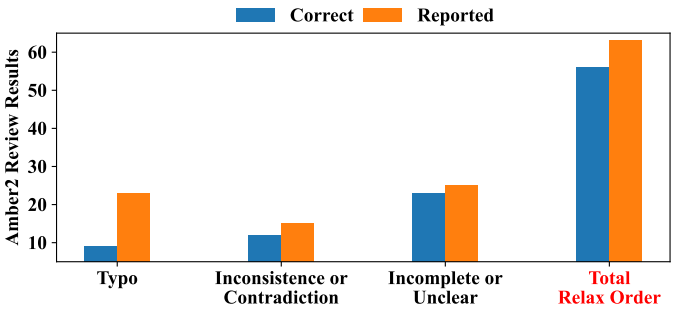

Fig. 11: The review results of Amber2.

of offering constructive advice. It can give the reasons why these types of errors exist and suggest ways to improve them.

D. Feedbacks Evaluation

Not all feedback provided by the LLM is correct or useful. Therefore, it is essential to have strategies in place to evaluate the review results of architecture specifications effectively. The most intuitive method is to ask the designers to check the outputs of the LLM. This approach, compared to directly reviewing the architecture specifications themselves, is already more efficient, potentially saving significant time and effort.

Figure 11 displays the review results for Amber2 based on the previously described prompts. Our analysis shows that the majority of the feedback provided by GPT is useful. When evaluating the responses across the entire specification file, the accuracy of the feedback is 88.89% (56 out of 63 responses). However, there are many instances where the responses do not match the specific defect type or chapter indicated by GPT. For example, in instances meant to report typographical errors, GPT may erroneously categorize issues related to inconsistence and contradiction, leading to a low accuracy for typographical errors in the review results. Furthermore, when reporting errors related to one chapter, GPT might wrongly include errors that occurred in other chapters. This problem can be alleviated when we split the specification file into individual chapters before reviewing. These two issues—mismatching defect types and incorrectly attributing errors to specific chapters—are the primary concerns in the responses generated by GPT. We also observe instances of feedback that 1) misinterpret sentences within the specification files, 2) lack context from subsequent chapters, and 3) misunderstand the information from figures. However, the occurrence of these issues is relatively rare.

VI. Conclusion

In this paper, we propose a novel framework for utilizing LLMs to generate and review architecture specifications. To facilitate our framework, we defined architecture specifications, categorized them into clear levels, and compiled a corresponding dataset. Our innovative methodology signifies a transformative step in architecture specification development, offering a path toward more efficient, accurate, and streamlined processes in chip design.

REFERENCES

[1] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023.

[2] S. Liu, W. Fang, Y. Lu, Q. Zhang, H. Zhang, and Z. Xie, "Rtlcoder: Outperforming gpt-3.5 in design rtl generation with our open-source dataset and lightweight solution," *arXiv preprint arXiv:2312.08617*, 2023.

[3] J. Blocklove, S. Garg, R. Karri, and H. Pearce, "Chip-chat: Challenges and opportunities in conversational hardware design," *arXiv preprint arXiv:2305.13243*, 2023.

[4] Y. Lu, S. Liu, Q. Zhang, and Z. Xie, "Rtllm: An open-source benchmark for design rtl generation with large language model," *arXiv preprint arXiv:2308.05345*, 2023.

[5] M. Liu, N. Pinckney, B. Khailany, and H. Ren, "Verilogeval: Evaluating large language models for verilog code generation," *arXiv preprint arXiv:2309.07544*, 2023.

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

[7] S. Thakur, J. Blocklove, H. Pearce, B. Tan, S. Garg, and R. Karri, "Autochip: Automating hdl generation using llm feedback," *arXiv preprint arXiv:2311.04887*, 2023.

[8] M. Nair, R. Sadhukhan, and D. Mukhopadhyay, "Generating secure hardware using chatgpt resistant to cwes," *Cryptology ePrint Archive*, 2023.

[9] M. Liu, T.-D. Ene, R. Kirby, C. Cheng, N. Pinckney, R. Liang, J. Alben, H. Anand, S. Banerjee, I. Bayraktaroglu *et al.*, "Chipnemo: Domain-adapted llms for chip design," *arXiv preprint arXiv:2311.00176*, 2023.

[10] Z. He, H. Wu, X. Zhang, X. Yao, S. Zheng, H. Zheng, and B. Yu, "Chateda: A large language model powered autonomous agent for eda," in *MLCAD Workshop*, 2023.

[11] Y. Fu, Y. Zhang, Z. Yu, S. Li, Z. Ye, C. Li, C. Wan, and Y. Lin, "Gpt4aigchip: Towards next-generation ai accelerator design automation via large language models," *arXiv preprint arXiv:2309.10730*, 2023.

[12] Z. Yan, Y. Qin, X. S. Hu, and Y. Shi, "On the viability of using llms for sw/hw co-design: An example in designing cim dnn accelerators," *arXiv preprint arXiv:2306.06923*, 2023.

[13] R. Kande, H. Pearce, B. Tan, B. Dolan-Gavitt, S. Thakur, R. Karri, and J. Rajendran, "Llm-assisted generation of hardware assertions," *arXiv preprint arXiv:2306.14027*, 2023.

[14] B. Ahmad, S. Thakur, B. Tan, R. Karri, and H. Pearce, "Fixing hardware security bugs with large language models," *arXiv preprint arXiv:2302.01215*, 2023.

[15] *Architecture Specification Dataset*, "https://anonymous.4open.science/r/SpecLLM-2247".

[16] *RISC-V Specifications*, "https://riscv.org/technical/specifications/".

[17] J. White, Q. Fu, S. Hays, M. Sandborn, C. Olea, H. Gilbert, A. Elnashar, J. Spencer-Smith, and D. C. Schmidt, "A prompt pattern catalog to enhance prompt engineering with chatgpt," *arXiv preprint arXiv:2302.11382*, 2023.