GenEDA: Towards Generative Netlist Functional Reasoning via Cross-Modal Circuit Encoder-Decoder Alignment

Wenji Fang, Wang Jing, Yao Lu, Shang Liu, Zhiyao Xie*
Hong Kong University of Science and Technology
*Corresponding Author

2025 IEEE/ACM International Conference On Computer Aided Design (ICCAD) | 979-8-3315-1560-7/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICCAD66269.2025.11240762

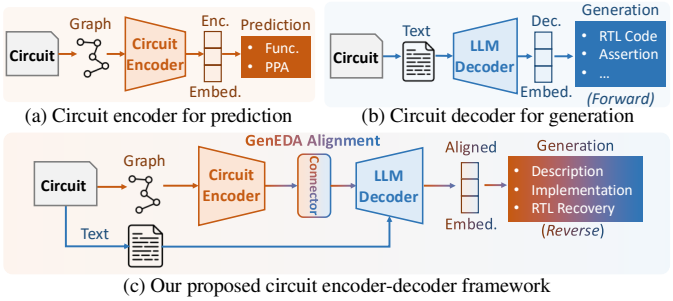
***Abstract*—The success of foundation AI has motivated the research of circuit foundation models, which are customized to assist the integrated circuit (IC) design process. However, existing pre-trained circuit foundation models are typically limited to standalone encoders for predictive tasks or decoders for generative tasks. These two model types are developed independently, operate on different circuit modalities, and reside in separate latent spaces. This restricts their ability to complement each other for more advanced capabilities. In this work, we present GenEDA, the first framework that cross-modally aligns circuit encoders with decoders within a shared latent space. GenEDA bridges the gap between graph-based circuit representation learning and text-based large language models (LLMs), enabling communication between their respective latent spaces. To achieve the alignment, we propose two paradigms to support both open-source trainable LLMs and commercial frozen LLMs. We leverage this aligned architecture to develop the first generative foundation model for netlists, unleashing LLMs' generative reasoning capability on the low-level and bit-blasted netlists. GenEDA enables three unprecedented generative netlist functional reasoning tasks, where it reversely generates high-level functionalities such as specifications and RTL code from low-level netlists. These tasks move beyond traditional gate function classification to direct generation of full-circuit functionality. Experiments demonstrate that GenEDA significantly boosts advanced LLMs' (e.g., GPT and DeepSeek series) performance in all tasks.**

I. Introduction

The ever-increasing IC complexity and skyrocketing IC design costs are challenging traditional electronic design automation (EDA) techniques. This trend has motivated the community's active exploration of new IC design methods, such as AI-assisted EDA techniques. Most recently, emerging *foundation AI models* have been customized and applied to IC design, named circuit foundation models [1], [2]. These pre-trained circuit foundation models target more generalized AI solutions for IC design.

Circuit foundation model: two main types. As Fig. 1 shows, existing circuit foundation models can be categorized into two main types: (a) circuit encoder for predictive tasks, and (b) circuit decoder for generative tasks. TABLE I provides a detailed comparison of these two types of works. **Circuit encoder** refers to pre-trained models that encode circuits into general embeddings (i.e., circuit representation learning). Taking these embeddings as input, lightweight downstream models are then fine-tuned for various *predictive* EDA tasks, such as circuit functional reasoning [3], [4], [5], [6], [7] and circuit quality prediction [8], [9], [10], [11]. **Circuit decoder** refers to pre-trained models with circuit-related *generative* capability. Built mostly on large language models (LLMs), these decoders generate text outputs, such as circuits in RTL code [12], [13], [14], verification assertions [15], [16], EDA tool scripts [17], etc.

Limitation: lack of alignment. Developed independently, circuit encoders and decoders operate on different modalities and handle circuit representations in clearly *distinct latent spaces*. Specifically: (a) Circuit encoders typically work in the circuit graph latent space. They excel at capturing circuit structural and functional properties into embeddings for predictive tasks, but lack generative capabilities. (b) Circuit decoders, typically LLMs, operate in the text latent space. They are effective at generating circuit-related text (e.g., RTL code, assertions), but they rely solely on textual input and cannot fully utilize the underlying structural information of circuits. As a result, these two important types are not aligned due to an inherent gap in their latent space, preventing more advanced capabilities.



(a) Circuit encoder for prediction (b) Circuit decoder for generation

(c) Our proposed circuit encoder-decoder framework

Fig. 1: (a) & (b) Two main types of existing circuit foundation models: encoders for prediction and decoders for generation. (c) *GenEDA* proposes a general framework that aligns circuit *encoders* with *decoders*, leveraging encoder-captured information to enhance decoder generation.

GenEDA: an encoder-decoder alignment framework. To address this, we present *GenEDA*[1], the first framework that aligns pre-trained circuit encoders with LLM decoders. GenEDA bridges these two major types by communicating *circuit graph latent space* and *text latent space*, enabling effective information exchange. This alignment allows structural and functional insights captured by circuit encoders to directly enhance the generative capabilities of LLM decoders.

GenEDA achieves the alignment by proposing two paradigms for both open-source trainable LLMs and commercial frozen LLMs: (1) Embedding-based alignment, which fine-tunes trainable LLMs using graph-based circuit embeddings by introducing a modality *connector*, and (2) Prediction-based alignment, which augments commercial frozen LLMs by feeding them textual predictions from the encoder.

Challenging application: generative functional reasoning of netlist. GenEDA's alignment framework can support unprecedentedly challenging generative applications, such as *generative* reasoning of the *netlist* functionality[2]. Unlike RTL code, netlists are composed of a huge number of low-level, bit-blasted gates and their complex connections, offering little human-readable semantics for LLMs to understand. While prior netlist encoders [7], [23], [24], [10], [19] can extract structural and functional features into embeddings, they are limited to predictive reasoning, only classifying the functionality of individual gates. GenEDA bridges this gap by aligning the encoder's structural and functional understanding of netlists with the decoder's generative strengths. This encoder-decoder alignment enables generating high-level functionality directly from low-level netlist inputs, which is an unprecedentedly challenging task due to the irreversible nature of logic synthesis.

[1]The code of GenEDA and benchmarks of proposed tasks are available at: https://github.com/hkust-zhiyao/GenEDA

[2]In the reverse netlist functional reasoning scenario, the high-level specification and RTL code as ground-truth are unknown to the model. Models are only provided with the low-level netlist as inputs.

Type	Method†	Input		Task			
		Format	Modality	Pred.	Gen.	Description	Direction‡
Encoder	DeepGate [18], etc.	AIG	Graph	✓		function pred.	Reverse
	CircuitFusion [8]	RTL	Graph & Text	✓		quality pred.	Forward
	MGVGA [9]	AIG		✓		func./quality pred.	Reverse/Forward
	NetTAG [19]	Netlist		✓		func./quality pred.	Reverse/Forward
Decoder	RTLCoder [20], etc.	Spec	Text (Image)		✓	RTL gen.	Forward
	HDLDebugger [21], etc.	RTL			✓	RTL debug	Forward
	AssertLLM [15], etc	Spec			✓	assertion gen.	Forward
	DeepRTL [22]⋆	RTL/Spec	Text		✓	RTL understand./gen.	Reverse/Forward
Enc-Dec	**GenEDA (ours)**	**Netlist**	**Graph&Text**	✓	✓	**Function gen.**	**Reverse**

† We list one of the representative works for each method in the table. For a more comprehensive comparison, please refer to Section II-A.
‡ Task direction is *forward* if they follow the VLSI design flow (e.g., predicting quality at the early stage, generating RTL from spec), and *reverse* if they go against it (e.g., predicting or generating function from netlist). Reverse tasks are challenging due to the design flow's irreversible nature.
⋆ This work [22] leverages T5, an encoder-decoder LLM. However, it targets only generative tasks, so we categorize it as a circuit decoder.

TABLE I: Comparison of GenEDA with representative categories of circuit foundation models. Existing circuit encoders mainly leverage graph structure for prediction tasks, while circuit decoders focus on semantic text for generation tasks. GenEDA bridges the gap between these two widely explored branches by aligning encoders and decoders within a shared latent space. After alignment, GenEDA supports more challenging generative netlist functional reasoning tasks.

Specifically, GenEDA reasons functionalities of given netlists in a wide spectrum of granularities, with outputs including: (1) general function description, (2) circuit implementation details, and (3) fine-grained exact RTL code. These GenEDA-supported new generative netlist functional reasoning tasks are highly valuable in multiple aspects: **(1) Practical applications:** Reasoning high-level functionality from bit-blasted netlists can support critical applications [25], [26], [23], such as functional verification, datapath optimization, and malicious logic detection. **(2) Unprecedented reasoning quality:** These tasks move beyond traditional gate function classification by directly generating the overall functionality of entire circuits, including specifications and RTL code, offering a significant leap in reasoning quality. **(3) Benchmarking model capability:** Our proposed tasks introduce new benchmarks for evaluating the generative circuit reasoning capabilities and netlist understanding of foundation models. Since these tasks generate human-readable circuit information, they help enhance the interpretability of circuit models.

The contributions of this paper are summarized as follows:

- **Circuit encoder-decoder alignment framework.** We propose the first framework that cross-modally aligns pre-trained circuit encoders with LLM decoders. It supports both trainable and frozen LLMs for advanced generative tasks through two alignment paradigms.
- **Generative netlist foundation model.** Leveraging this framework, we develop the first generative foundation model for netlists. By integrating structural and functional insights captured by netlist encoders, the model unleashes LLMs' generative reasoning capability on the low-level and bit-blasted netlists.
- **New generative netlist reasoning tasks and benchmarks.** We propose three novel generative netlist functional reasoning tasks with corresponding benchmarks, advancing beyond prior gate function classification. We also release these benchmarks to encourage follow-up research on these tasks.
- **Boosting SOTA LLMs' performance.** Experimental results validate that GenEDA significantly boosts the performance of cutting-edge LLMs on all three new functional reasoning tasks after alignment with the pre-trained netlist encoder.

II. Related Work

A. Method Related: Circuit Foundation Model

Recent advances in foundation AI for EDA have enabled strong generalization and generation capabilities through the *pretrain-finetune* process. As summarized in TABLE I, these circuit foundation models can be categorized into encoder-based and decoder-based architectures, each supporting different inputs and tasks.

Circuit encoders for prediction. Encoder-based models typically learn structured circuit representations to support predictive tasks such as reverse functional reasoning and early-stage design quality estimation. Most methods [6], [5], [4], [27], [18], [7], [24], [10], [28] focus on AIG netlists and use graph learning to capture the circuit structure. Recent work [8], [9], [19] fuses multiple modalities (graph, text, image), but their multimodal fusion are limited to the encoder side, lacking alignment with LLMs for enabling generation capabilities. **Circuit decoders for generation.** LLM-based decoders support forward-generation tasks like RTL or assertion generation [29], [20], [30], [15], [31], debugging [21], optimization [13], [32], and knowledge querying [33], [34]. DeepRTL [22] further explores bidirectional generation between RTL and specification. However, since RTL is easier for LLMs to understand due to its rich semantics, the text-only T5 LLM is sufficient for these tasks. Additionally, existing multimodal approaches focus on circuit images [32], [15], a capabiltity already present in current multimodal LLMs. They overlook the crucial circuit graph structure, which remains largely unexplored in combination with LLMs.

Circuit encoder-decoder alignment by GenEDA. GenEDA bridges this gap by proposing the first cross-modal encoder-decoder alignment framework, which aligns circuit structural representations with LLM textual representations in a shared latent space, enabling support for advanced generative tasks.

B. Application Related: Functional Reasoning on Netlists

Functional reasoning on gate-level netlists aims to reconstruct the high-level functionality originally described in specifications or RTL.

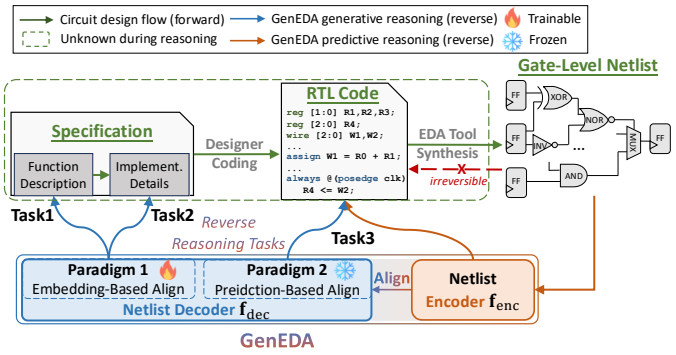


Fig. 2: Overview of GenEDA framework integrated into the standard IC design flow. GenEDA aligns the pre-trained netlist encoder with LLM decoders through two alignment paradigms, enabling challenging generative netlist functional reasoning tasks.

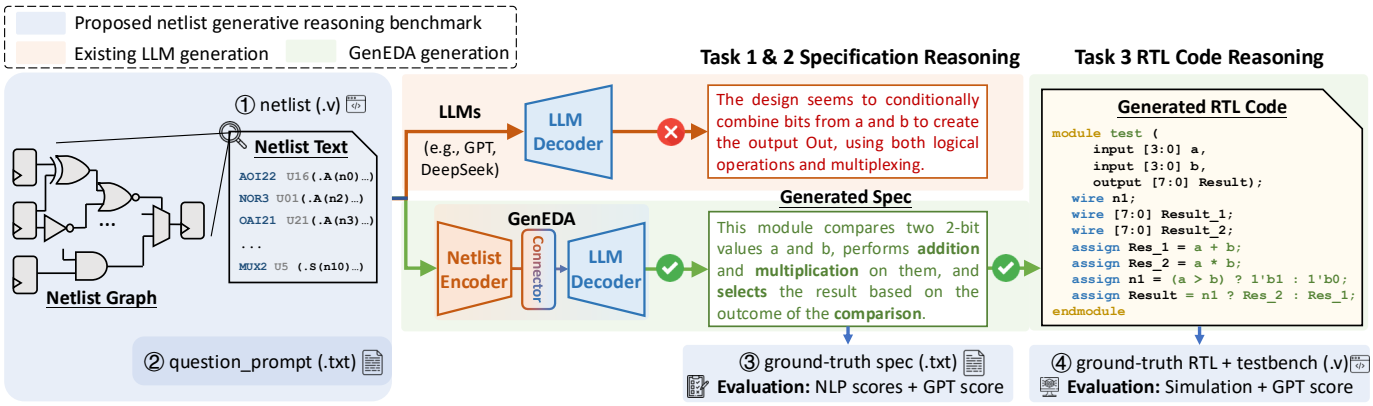


Fig. 3: Proposed generative netlist functional reasoning benchmarks. For Tasks 1 and 2, the netlist and question prompt are processed by models for specification reasoning, and evaluated using NLP scores and GPT scores. For Task 3, the models reconstruct RTL code from the netlist, which is evaluated via simulation and GPT scores.

It plays a critical role in functional verification, logic optimization, datapath synthesis, and hardware security. Existing approaches primarily fall into two categories: formal methods for analyzing functionality [25], [35], [26], [36], and machine learning methods for gate-level function classification [37], [38], [7], [23], [10]. We detail these two categories below.

Formal analysis. Traditionally, netlist functional reasoning relies on structural and functional analysis using formal techniques [25], [35], [26], [36]. These approaches typically extract subcircuits from the netlist and match them against components in a golden library via exhaustive formal verification. However, they are time-consuming, dependent on library completeness, and incapable of recognizing functional variants. **GNN-based individual gate function *prediction*.** Recently, GNN-based methods [37], [38], [7], [23], [10] have been applied to predictive netlist functional reasoning. These methods focus on learning the netlist structure for gate-level function classification. While they effectively identify the roles of known components, they are limited in generalizing to unseen functionality and cannot reason about the full behavior of the entire circuit.

GenEDA-based entire circuit function *generation*. GenEDA advances from gate-level prediction to full-circuit generative reasoning. By aligning netlist encoders with LLM decoders, it enables the direct generation of high-level specifications and RTL code from low-level netlists.

III. Overview

Fig. 2 presents the overview of our GenEDA framework, integrated into the standard digital IC design flow. GenEDA aligns the state-of-the-art post-synthesis netlist encoder NetTAG [19] $\mathbf{f}_{\text{enc}}$ with cutting-edge LLM-based decoders $\mathbf{f}_{\text{dec}}$. It first converts the circuit netlist $\mathcal{N}$ into embeddings via the encoder $\mathbf{f}_{\text{enc}}$, capturing both netlist structural and functional information. The encoder output is then provided to aligned decoders $\mathbf{f}_{\text{dec}}$ to support advanced generative reasoning tasks on netlists.

We propose two novel encoder-decoder alignment paradigms for both open-source trainable LLMs and commercial frozen LLMs: (1) Embedding-based alignment, where a trainable LLM decoder is instruction-tuned with encoder embeddings with a circuit modality connector. (2) Prediction-based alignment, where the encoder annotates textual gate function predictions on netlists to augment the inputs to the frozen LLM. In our experiments, we apply paradigm 1 for specification reasoning tasks (Task 1 and Task 2) and paradigm 2 for exact RTL code reasoning (Task 3).

The rest of the paper is organized as follows: In Section IV, we first provide a detailed explanation of our three novel generative netlist functional reasoning tasks and our contributed benchmarks. In Section V, we illustrate the two proposed encoder-decoder alignment paradigms, as well as how to tackle the three tasks. In Section VI, we demonstrate the effectiveness of GenEDA in all three generative functional reasoning tasks in experiments. Finally, in Section VII, we discuss the potential of extending GenEDA to other circuit design stages and the broader applications of generative netlist functional reasoning.

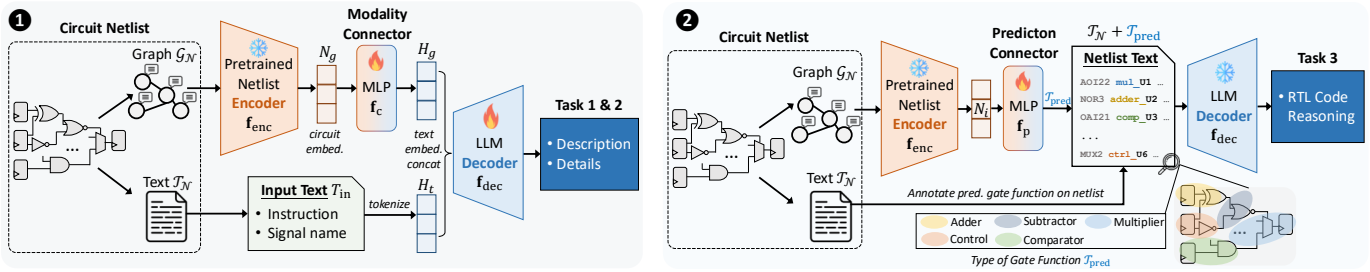
IV. Benchmarking Generative Functional Reasoning

Fig. 3 illustrates our three novel generative functional reasoning tasks on netlists, along with the benchmarks developed to support them. These tasks aim to reversely generate high-level circuit functionality, including natural language specifications and exact RTL code, from low-level bit-blasted netlists. Please note that models are only provided with netlists, their corresponding specifications and RTL code serve as ground-truths and and unknown to the model. Our proposed benchmarks evaluate the generative model's ability to truly understand the functionality of netlists, setting a new direction for generative EDA tasks. We detail the three tasks below.

A. Task 1 & 2: specification reasoning from netlist.

Task and benchmark description. Tasks 1 and 2 aim to reversely generate high-level natural language specifications from gate-level netlists, as shown in Fig. 3. **Task 1 Function description generation** focuses on generating circuit functional descriptions from low-level netlists, emphasizing the overall behavior of the circuit. **Task 2 Implementation detail generation** targets the reconstruction of step-by-step signal propagation and logic behavior from the netlist, reflecting the underlying design implementation. As these are novel tasks with no prior benchmarks, we construct new datasets and benchmarks to support model training and evaluation. Specifically, we collect 400 circuit netlists with various design scales and complexities, annotated with natural language specifications as ground-truth. For each design, our proposed benchmark provides the following information in three separate files:

- **Netlist text.** Gate-level netlist in Verilog text format synthesized from RTL code. Please note that in these reverse tasks, the RTL code is unknown to the model.
- **Question prompt.** For Task 1, the prompt inquires models to describe the interface, purpose, functionality, and constraints of the netlist. For Task 2, the prompt asks the model to explain the combinational logic, sequential behavior, and control flow.
- **Ground-truth specification text.** Since real-world RTL specifications are rarely available, following [22], [8], we generate ground-truth reference answers for these two tasks using GPT-4o prompted with RTL code and manually verify their quality through expert review.



(a) Paradigm 1: Embedding-based alignment

(b) Paradigm 2: Prediction-based alignment

Fig. 4: Two encoder-decoder alignment paradigms in GenEDA: 1. Embedding-based alignment integrating embeddings from the netlist encoder with trainable LLM decoders, and 2. Prediction-based alignment using functional predictions from the netlist encoder as textual input for frozen LLM decoders.

Evaluation metrics. As there are no standard metrics for evaluating the functional similarity of specification texts, we follow prior work DeepRTL [22] (i.e., generating specification from RTL code) to adopt a combination of natural language metrics. Specifically, we use BLEU, ROUGE-1/2/L, and cosine similarity computed from text embeddings. Additionally, GPT-4o is employed as an automated evaluator to assess semantic similarity between generated outputs and reference specifications. The prompt templates used are provided in Section VI.

B. Task 3: arithmetic RTL code reasoning from netlist.

Task and benchmark description. Task 3 arithmetic RTL code reasoning targets the reverse generation of RTL code from gate-level netlists, specifically for arithmetic circuits, as shown in Fig. 3. Unlike existing *forward* RTL code generation benchmarks [39], [40], which rely on human-readable specifications as input, our *reverse* task begins with post-synthesis netlists. This makes the task significantly more challenging due to the irreversible nature of logic synthesis and the lack of high-level functional information.

We propose the first benchmark for reverse RTL code reasoning. Given the task's complexity, we begin by focusing on arithmetic modules. Specifically, we extend the GNN-RE gate-level arithmetic function prediction dataset [37] into a generative benchmark, incorporating golden RTL code and testbenches. For each of the 9 arithmetic designs in our benchmark, we provide:

- **Netlist text.** Gate-level netlist from GNN-RE dataset [37], originally used for gate function prediction task.
- **Question prompt.** Instructions to first infer the word-level arithmetic function, and then implement it using RTL code.
- **Ground-truth RTL code.** The original RTL design used for synthesis, which serves as the reconstruction target.
- **Testbench.** A verified testbench with pre-defined module name and IO ports, containing multiple input-output cases for functional validation.

Evaluation metrics. Similar to the existing forward RTL generation evaluation metrics [39], [40], we validate both syntax correctness and function correctness of the reversely generated arithmetic RTL code using our provided golden testbenches. Additionally, GPT-4o is employed to assign a function similarity score, measuring how closely the generated RTL matches the ground truth. We provide detailed evaluation method implementation and prompts for obtaining GPT-score in Section VI.

V. GenEDA Encoder-Decoder Framework

Fig. 4 presents our proposed two paradigms for the encoder-decoder alignment, accommodating both trainable open-source LLMs and frozen commercial LLMs. Through alignment, GenEDA integrates rich netlist information captured by the encoder to enhance the generative reasoning capabilities of decoder LLMs. These two paradigms with their supported tasks are introduced in Section V-A and V-B, respectively. In our experiments, Paradigm 1 is applied to Tasks 1 and 2 for specification reasoning, while Paradigm 2 is used for Task 3 to generate exact RTL code.

Netlist encoder and LLM decoder in GenEDA. Before discussing the alignment mechanisms, we briefly introduce the models used in GenEDA. For the encoder, we employ NetTAG [19], the state-of-the-art encoder capable of handling post-synthesis netlists, whereas most prior netlist encoders are limited to the AIG format. NetTAG introduces a text-attributed graph format for netlists and employs a multimodal architecture: it encodes the gate function via symbolic Boolean expressions using an LLM encoder and captures global circuit structure through a graph transformer. This results in rich embeddings that encode both functional and structural information. On the decoder side, we consider both general-purpose LLMs, such as OpenAI GPT and DeepSeek, and circuit-specific LLMs like RTLCoder [20], which is fine-tuned for forward RTL code generation.

A. Paradigm 1: Embedding-Based Alignment

Paradigm ❶ overview. Fig. 4 (a) shows our paradigm 1, which aligns trainable open-source LLM decoders $\mathbf{f}_{dec}$ with the embeddings generated by the netlist encoder $\mathbf{f}_{enc}$ from the netlist graph (i.e., text-attributed graph) modality. The main challenge is the modality gap between the encoder-generated netlist embeddings and the text embeddings expected by the decoder, as they lie in fundamentally different latent spaces. To address this, our paradigm 1 introduces a modality connector $\mathbf{f}_c$, which acts as a "circuit tokenizer", transforming the circuit embeddings from our encoder into text-alike embeddings compatible with the LLM decoder. The connector $\mathbf{f}_c$ is cross-modally instruction-tuned with the LLM $\mathbf{f}_{dec}$, enabling deep embedding-level alignment in the shared latent space. We describe our aligned model architecture and training method below.

Embedding alignment architecture. As shown in Fig. 4 (a), given an input netlist $\mathcal{N}$, we represent it in two modalities: text-attributed graph $\mathcal{G}_\mathcal{N}$ and netlist Verilog code $\mathcal{T}_\mathcal{N}$ in text format. For $\mathcal{G}_\mathcal{N}$, we employ our pre-trained netlist encoder $\mathbf{f}_{enc}$ to process the netlist and generate the netlist graph-level embedding N_g, which contains the structural and functional netlist information. For the netlist Verilog text $\mathcal{T}_\mathcal{N}$, we keep only the signal names and leave the structural details to be captured by the encoder model. As shown in Fig. 5, the signal name text is combined with the question prompt instruction to form the textual input $\mathcal{T}_{in}$. This input $\mathcal{T}_{in}$ is then tokenized into language embeddings H_t and fed into the LLM decoder $\mathbf{f}_{dec}$.

To align the encoded embeddings N_g with the decoder $\mathbf{f}_{dec}$ embedding H_t, we introduce a trainable connector MLP $\mathbf{f}_c$ to transform N_g into language-modal embedding tokens H_g. These converted embeddings have the same dimension as the word embedding (i.e.,

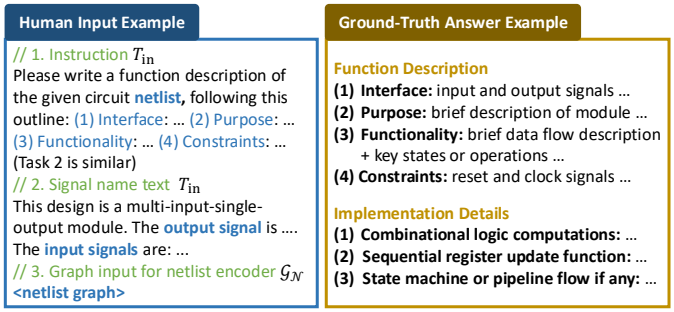


Fig. 5: Instruction tuning data pair of alignment paradigm 1.

H_t) space in the trainable LLM decoder $\mathbf{f}_{\text{dec}}$:

$$\begin{aligned} &\mathbf{f}_{\text{dec}}(H_g, H_t) \Rightarrow \text{Generative Reasoning,} \\ &\text{with } H_g = \mathbf{f}_{\text{c}}(N_{\text{g}}) \text{ and } N_{\text{g}} = \mathbf{f}_{\text{enc}}(\mathcal{G}_{\mathcal{N}}). \end{aligned} \tag{1}$$

Training for encoder-decoder alignment. To enhance embedding alignment, we propose cross-modal instruction tuning using task-specific datasets (i.e., from Task 1 & 2) to achieve encoder-decoder alignment. As shown in Fig. 5, we generate a multimodal instruction-response pair for each netlist, where the input includes: (1) A task-specific instruction (e.g., request for function description or implementation details). (2) Input and output signals extracted from the netlist code. (3) The netlist graph format, with the encoder capturing the structural and functional information. The ground truth is the golden specification according to the task. This creates a unified format for multimodal instruction-following sequences. GenEDA is instruction-tuned on prediction tokens using the auto-regressive training objective, maximizing the likelihood of the target ground-truth specification text sequence y with length L, formulated as:

$$\mathcal{L}_{\text{align1}} = -\sum_{i=1}^{L} \log p(y_i \mid y_{<i}, \mathcal{G}_{\mathcal{N}}) \tag{2}$$

where $y_{<i}$ represents the previously generated tokens before the current token y_i, and $\mathcal{G}_{\mathcal{N}}$ denotes the input netlist graph for the encoder.

During instruction tuning, we leverage a two-step procedure for multimodal embedding alignment:

1) *Pre-training for modality alignment.* In this stage, the netlist encoder $\mathbf{f}_{\text{enc}}$ and decoder $\mathbf{f}_{\text{dec}}$ remain frozen. Only the connector MLP $\mathbf{f}_{\text{c}}$ is trained to maximize the likelihood of the generated tokens of the auto-regressive loss, as formulated in Equation (2). This step aligns the netlist embeddings H_g with the pre-trained LLM word embeddings H_t, effectively acting as a modality adapter (i.e., "netlist tokenizer") for the LLM.
2) *Fine-tuning end-to-end.* In this stage, the encoder $\mathbf{f}_{\text{enc}}$ remains frozen, while both the connector $\mathbf{f}_{\text{c}}$ and the LLM decoder $\mathbf{f}_{\text{dec}}$ are fine-tuned also with the auto-regressive loss (i.e., Equation (2)). This end-to-end training step further refines the alignment, allowing the model to perform generative tasks seamlessly across graph and text modalities.

B. Paradigm 2: Prediction-Based Alignment

Paradigm ❷ overview. In addition to trainable open-source LLMs, advanced LLMs are often frozen due to commercial or computational limitations, yet they excel in reasoning and support longer input contexts. Unlike paradigm 1, which aligns coarse-grained graph-level embeddings N_g, our paradigm 2 leverages fine-grained gate-level text and frozen advanced LLMs to support the more challenging exact RTL code reasoning task. As Fig. 4 (b) shows, we align our multi-modal encoder $\mathbf{f}_{\text{enc}}$ with frozen LLM decoders $\mathbf{f}_{\text{dec}}$ by using the encoder's fine-grained gate functional predictions $\mathcal{T}_{\text{pred}}$ as textual inputs for LLMs. These predictions $\mathcal{T}_{\text{pred}}$ provide detailed gate-level analysis of netlist functionalities, which are then used by LLMs to summarize and generate high-level functionality. This paradigm enables seamless integration without modifying the pre-trained frozen decoder. Below, we introduce the details of the task and our alignment method.

Prediction alignment architecture. To generate functional predictions $\mathcal{T}_{\text{pred}}$ with the encoder $\mathbf{f}_{\text{enc}}$, we fine-tune our pre-trained encoder to enable gate functionality identification with the task proposed in [37]. This task involves classifying gates into 5 high-level function types defined in their arithmetic RTL code, including adder, multiplier, comparator, subtractor, and controller. During fine-tuning, the encoder $\mathbf{f}_{\text{enc}}$ remains frozen, and only the additional function predictor MLP $\mathbf{f}_{\text{p}}$ is trained for gate function classification using cross-entropy loss, as formulated below.

$$\mathcal{L}_{\text{align2\_pred}} = -\sum_{i} y_i \log(\mathbf{f}_{\text{p}}(N_i)), \tag{3}$$

where y_i represents the ground-truth function type label for the netlist gate i, and N_i are the encoder-generated embeddings for this gate.

After fine-tuning, the encoder generates textual predictions $\mathcal{T}_{\text{pred}}$ for all gates' functionalities. These predictions (i.e., gate functional roles) are directly annotated into its netlist Verilog code $\mathcal{T}_{\mathcal{N}}$, which serves as input text for the frozen LLM decoder. The decoder then utilizes this fine-grained gate annotation for arithmetic RTL code generation. To illustrate this process more clearly, we present a detailed case study in Fig. 8. We formulate the process below:

$$\begin{aligned} &\mathbf{f}_{\text{dec}}(\mathcal{T}_{\text{pred}} + \mathcal{T}_{\mathcal{N}}) \Rightarrow \text{Generative Reasoning,} \\ &\text{with } \mathcal{T}_{\text{pred}} = \mathbf{f}_{\text{p}}(\mathbf{f}_{\text{enc}}(\mathcal{G}_{\mathcal{N}})). \end{aligned} \tag{4}$$

Chain-of-Thought for RTL code reasoning. Leveraging the annotated netlist, we employ the Chain-of-Thought (CoT) technique to improve the LLM in this challenging exact RTL code reasoning. The CoT prompt decomposes reasoning tasks into two sequential steps: (1) Reason about the word-level arithmetic function description based on the given circuit netlist with gate annotations. (2) Generate RTL code to implement the identified arithmetic function, ensuring word-level RTL operations and avoiding bit-level operations.

VI. Experiments

In this section, we first introduce the experimental setup and evaluation metrics in Section VI-A. Then, in Section VI-C, we present our results on specification context reasoning (i.e., Task 1 & 2). Then we discuss the arithmetic RTL code reasoning results (i.e., Task 3) in Section VI-D and conclude with the ablation study in Section VI-E.

A. Experimental Settings

Circuit dataset preparation. For the circuit encoder and Task 1 & 2, we collect circuit datasets from various open-source RTL code benchmarks, including ITC99 [41], OpenCores [42], Chipyard [43], and VexRiscv [44]. In Task 1 and 2, for large sequential circuits, we split them into multiple sub-circuits, following the techniques in [19]. After splitting, we collect 25k subcircuits, which are augmented by functional equivalent transformation by Yosys [45] to create a total

TABLE II: Statistics of the netlist dataset.

	Source	# Circuits	# Tokens (avg.)	# Gates (avg.)
Task 1 & 2	ITC99 [41]	4k	15k	1025
	OpenCores [42]	55k	9k	173
	Chipyard [43]	20k	24k	2813
	VexRiscv [44]	21k	13k	901
Task 3	GNN-RE [37]	8	4k	67

TABLE III: Evaluation results on Task 1 & 2, reasoning specification text from gate-level netlists. Best results are highlighted in bold.

Model	Task 1 Functional Description Reasoning						Task 2 Implementation Detail Reasoning					
	BLEU	ROUGE-1	ROUGE-2	ROUGE-L	Emb. Sim.	GPT Score	BLEU	ROUGE-1	ROUGE-2	ROUGE-L	Emb. Sim.	GPT Score
GPT4o	5	34	10	17	0.83	0.21	5	34	10	17	0.84	0.44
DeepSeek-V3	4	31	9	17	0.82	0.19	3	34	10	17	0.83	0.42
DeepSeek-1B	0	7	1	5	0.77	0.04	0	5	1	4	0.73	0.01
DeepSeek-7B	0	8	2	6	0.77	0.06	0	5	1	4	0.74	0.02
RTLCoder-7B*	0	5	1	4	0.32	0.02	0	3	1	2	0.28	0.01
GenEDA-1B	12	47	16	27	0.9	0.62	12	47	19	28	0.93	0.5
GenEDA-7B	**14**	**49**	**19**	**28**	**0.91**	**0.62**	**14**	**50**	**21**	**30**	**0.94**	**0.51**

* RTLCoder [20] is fine-tuned for generating RTL code from specifications. In our setting, we adopt the fine-tuned model as a circuit-specific LLM baseline to generate specifications from netlists reversely.

Task 1 Response Example

(1) Interface: The module has numerous input signals including control and data inputs from various **csr** (Control and Status Register) and **pmp** (Physical Memory Protection) .... The output signal is `**csr_reg_pmp_3_addr**`.
(2) Purpose: The purpose of this module is to **manage control and status registers (CSR), handle memory protection**, and update outputs based on these operations.
(3) Functionality: The module reads various CSR and PMP configurations, performs logic operations, and updates the output based on the logic. Key states or operations include: - **CSR read and write operations** - **Memory protection configuration** - **Address decoding and control signal generation** - Conditional data assignment based on specific conditions.
(4) Constraints: The module is synchronized with a **clock signal**, and it can be reset using a **reset signal**.

Ground Truth Specification

(1) Interface: - Inputs: clk, rst, **csr**, **pmp**, ... - Outputs: **csr_reg_pmp_3_addr**
(2) Purpose: - This module seems to **handle a variety of Control and Status Registers (CSRs)** within a system, likely part of a RISC-V processor or similar architecture. It manages read/write operations with CSR related computations and control flow.
(3) Functionality: - The module coordinates interactions with a wide array of **CSR registers** to determine their **read and write states**. Primarily, it seems to decide whether CSRs can be accessed based on the current CSR commands and inputs (likely reflecting different privilege modes and states of the processor). Key operations include decoding instruction inputs to **access specific registers**, handling of interrupts, and **managing PMP protections**.
(4) Constraints: - The design uses **reset** and **system clock** for its sequential logic execution.

Fig. 6: Case study for Task 1. Comparison between the response from GenEDA and the ground truth specification for a circuit module.

of 50k samples. We randomly sample 400 subcircuits for testing, ensuring that no subcircuits from the same circuit are included in the training set. Since no specification documents are available, we generate functional specifications from the RTL context using GPT-4o as the ground-truth for instruction-tuning. Human engineers then verify these generated specifications, and those that do not contain valid functionality are excluded. For Task 3, we use the open-source arithmetic RTL code from [37]. Please note that in all tasks, only low-level netlists are provided for reverse reasoning, with high-level specifications and RTL code being unknown to models. All RTL designs are synthesized into netlists using the Synopsys Design Compiler with the NanGate 45nm technology library. We provide detailed statistics of our netlist dataset in TABLE II, including the number of circuits, the average number of text tokens, and the average number of gates.

Model and training. For the encoder model in GenEDA, we employ the state-of-the-art netlist encoder NetTAG [19] as the backbone. For the decoder LLMs in GenEDA, we fine-tune the DeepSeek-Coder [46] 1B and 7B models in the trainable embedding-based alignment (Paradigm 1), and directly leverage the commercial APIs of OpenAI GPT-4o and DeepSeek-V3 [47] as the frozen LLMs in the prediction-based alignment (Paradigm 2). The training process utilizes DeepSpeed ZeRO and LoRA techniques. In alignment paradigm 1, we adopt a three-layer MLP with dimensions 768, 2048, and 4096 to transform the netlist embedding (768 dimensions) into the LLM word embeddings (4096 dimensions). This connector can be further explored using more advanced methods, such as Q-Former and cross-attention mechanism as in [48], [49], [50][3]. GenEDA is instruction-tuned using LoRA on the full task-specific dataset for one epoch. In alignment paradigm 2, the function predictor MLP for encoder fine-tuning contains three layers with a hidden dimension of 256. Experiments are conducted on a server equipped with 8 NVIDIA A800 80G PCIe GPUs.

SPEC Similarity Evaluation by GPT

- You are a professional Verilog designer that needs to evaluate the similarity between two textual functional summaries describing VLSI designs.
- The first summary is the ground truth description of the circuit, and the second summary is the generated description of the circuit.
- Please read the following summaries and provide a similarity score between 0 and 1 based on how similar the two summaries are in terms of describing the functionality of the designs, where 0 means completely dissimilar and 1 means identical. Note that you should strictly only output the score without any additional information.

RTL Code Similarity Evaluation by GPT

- You are a professional Verilog designer that needs to evaluate the functional similarity between two VLSI designs in Verilog code format.
- The first Verilog is the ground truth, and the second one is the generated Verilog of the circuit.
- Please first analyze their implemented arithmetic functionality and provide a similarity score between 0 and 1 based on how similar the two Verilog code are in terms implemented arithmetic, where 0 means completely dissimilar and 1 means their implemented functionalities are identical. Note that you should strictly only output the score without any additional information.

Fig. 7: Prompt used in GPT-assisted evaluation (i.e., GPT Score).

B. Benchmark Evaluation

For specification reasoning tasks (Task 1 & 2), the model-generated natural language specification is compared directly with the ground-truth specification using both natural language similarity metrics and LLM-assisted functionality evaluation metrics. Specifically, natural language similarity scores, including BLEU, ROUGE-1/2/L, are computed. These metrics assess the overlap between the generated text and the reference text by comparing n-grams. For LLM-based evaluation, we use the OpenAI text embedding model, text-embedding-ada-002, to obtain text embeddings for both the generated and reference specifications. Cosine similarity is then calculated between the embeddings to generate the embedding similarity score. Additionally, we utilize GPT-4o to directly evaluate the specifications and assign a similarity score between 0 and 1 based on how closely the generated specification matches the intended functionality of the designs. Detailed prompts for GPT-4o are provided in Fig. 7.

For RTL code reasoning (Task 3), we use Synopsys VCS to simulate the generated RTL code with the proposed testbench, validating both syntax and functional correctness. Each circuit is generated five times, and we compute the average success rate and Pass@k metrics [20]. Similar to specification reasoning, GPT-4o is also used to evaluate the functional similarity between the generated and ground-truth RTL codes, with the prompt outlined in Fig. 7.

[3]Complex connectors like Q-Former are typically used with frozen open-source LLMs [49], [50], while simpler connectors such as MLPs are often suitable when the LLM is trainable [48].

TABLE IV: Evaluation results on Task 3, reasoning arithmetic RTL code from gate-level netlists. Each design is generated five times per model. Best results are highlighted in bold.

	GPT-4o			GenEDA (w. GPT-4o)			DeepSeek-V3			GenEDA (w. DeepSeek-V3)		
Circuit	Syntax	GPT Score	Function	Syntax	GPT Score	Function	Syntax	GPT Score	Function	Syntax	GPT Score	Function
1	80%	0.36	0%	100%	0.52	20%	100%	0.18	0%	100%	0.85	80%
2	20%	0.32	0%	100%	0.88	80%	100%	0.55	40%	100%	0.99	100%
3	100%	0.3	60%	100%	0.28	40%	100%	0.2	0%	100%	0.74	60%
4	100%	0.66	60%	40%	0.44	20%	100%	0.95	100%	100%	1	100%
5	60%	0.18	0%	80%	0.51	0%	100%	0	0%	100%	0.98	80%
6	20%	0.2	0%	100%	0.79	60%	100%	0.28	0%	100%	0.95	100%
7	80%	0.18	0%	80%	0.54	0%	60%	0.18	0%	80%	0.91	0%
8	40%	0.28	0%	80%	0.7	0%	40%	0.49	0%	80%	0.76	0%
9	80%	0.3	0%	100%	0.56	40%	100%	0.2	0%	100%	0.78	100%
Success Rate	64%	0.31	13%	87%	0.58	29%	89%	0.34	16%	**96%**	**0.88**	**69%**
Pass@1	67%	/	17%	83%	/	50%	92%	/	8%	**100%**	/	**58%**
Pass@5	100%	/	22%	100%	/	67%	100%	/	22%	**100%**	/	**78%**

Task 3 CoT Prompt

You are a professional Verilog designer.
Step 1: Please first reason the word-level arithmetic function of the given circuit netlist with gate functionality annotations.
Step 2: Then, follow the arithmetic function to write an RTL code to implement the functionality. Please use word-level operations, avoiding bit-level operations.
Here is the netlist Verilog code with gate annotations:

```
module test ( a, b, Result );
  input [0:3] a;
  input [0:3] b;
  output [0:7] Result;
  XNOR3 adder_U4 (.A(b[0]),.B(a[0]),.C(n1),
    .Y(Result_1[2]));
  NAND2 adder_U3 (.A(a[1]),.B(b[1]),.Y(n1));
  OA21 adder_U2 (.A0(a[1]),.A1(b[1]),.B0(n1),
    .Y(Result_1[3]));
  AND3 multiplier_U6 (.A(a[0]),.B(b[0]),
    .C(Result_2[3]),.Y(Result_2[0]));
  .....
  AOI22 multiplier_U1 (.A0(a[0]),.A1(b[1]),
    .B0(b[0]),.B1(a[1]),.Y(n2));
  INV comparator_U2 (.A(a[0]),.Y(n1));
  AO21 comparator_U1 (.A0(b[0]),.A1(n1),
    .B0N(a[1]),.Y(n2));
endmodule
```

Task 3 Response from GenEDA

Step 1: Reasoning the Word-Level Arithmetic Circuit Function
The given Verilog netlist describes a module `test` that takes two **4-bit inputs `a` and `b`** and produces an **8-bit output `Result`**. The module consists of three main components:
1. Adder: This component adds the two 4-bit inputs `a` and `b` and produces an 8-bit result `Result_1`.
2. Multiplier: This component multiplies the two 4-bit inputs `a` and `b` and produces an 8-bit result `Result_2`.
3. Comparator: This component compares the two 4-bit inputs `a` and `b` and produces a 1-bit output `n1`, which is `1` if `a > b` and `0` otherwise. The final **output `Result`** is determined based on the value of `n1`:
- If `n1` is `1`, the output `Result` is selected from `Result_2`.
- If `n1` is `0`, the output `Result` is selected from `Result_1`.

Step 2: Writing the RTL Code

```
module test (
        input [3:0] a,
        input [3:0] b,
        output [7:0] Result);
  wire n1;
  wire [7:0] Result_1;
  wire [7:0] Result_2;
  assign Result_1 = a + b;
  assign Result_2 = a * b;
  assign n1 = (a > b) ? 1'b1 : 1'b0;
  assign Result = n1 ? Result_2 : Result_1;
endmodule
```

Task 3 Response from GPT-4o

```
module test (
  ...
  assign Result_1[4] = a[0] ^ b[0] ^ (a[1] & b[1]);
  assign Result_1[5] = a[1] + b[1] + (a[2] & b[2]);
  assign Result_1[7] = ~(a[3] | b[3]);
  assign Result_2[4] = a[3] ^ b[3];
  assign Result_2[6] = a[1] & b[1];
  ...
  assign n1 = (a > b) ? 1'b1 : 1'b0;
  assign Result[0] = n1 ? Result_2[0] : Result_1[0];
endmodule
```

Fig. 8: Case study for Task 3. By utilizing gate function predictions from GenEDA's encoder, the LLM in GenEDA can reason about the *word-level* arithmetic functionality of bit-blasted netlists and accurately reconstruct the corresponding RTL code (blue box). In contrast, without these predictions, LLMs (e.g., GPT-4o) generate only *bit-level* operations (red box), leading to significantly lower reconstruction accuracy.

C. Result of Specification Reasoning (Task 1 & 2)

Baseline solutions. To ensure a comprehensive comparison, we evaluate both general-purpose and circuit-adapted LLMs as baseline methods. For general-purpose LLMs, we use advanced commercial models like GPT-4o and DeepSeek-V3, as well as lightweight open-source models like DeepSeek-Coder 1B and 7B. For circuit-adapted LLMs, we select RTLCoder-7B [20], a representative LLM fine-tuned for spec-to-RTL generation. These baselines take netlist text and the question prompt as input and generate the specification text.

Comparison with baselines. TABLE III presents the evaluation results for specification reasoning in Task 1 and Task 2, comparing GenEDA models with baseline models. For both tasks, after aligning with netlist encoder embeddings through multimodal instruction tuning, our GenEDA models (1B and 7B), based on DeepSeek Coder 1B and 7B, significantly outperform all general-purpose LLMs across all textual semantic similarity metrics. Additionally, although RTLCoder [20] is fine-tuned for RTL code generation from specifications, it performs poorly on both tasks, even underperforming its base model, DeepSeek-7B. This is primarily due to the substantial differences between the tasks. Notably, for the GPT scores, which analyze the similarity between generated specifications and ground truth with GPT, GenEDA scored much higher than the baseline LLMs.

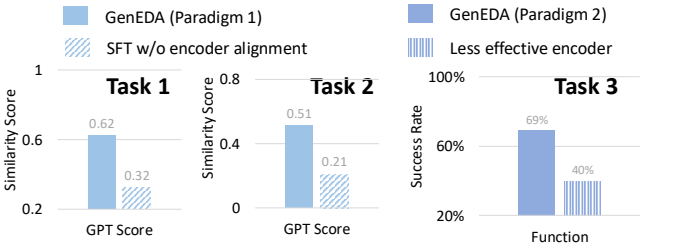
These results highlight the effectiveness of aligning circuit structural and functional information through encoders to enhance generative capabilities. Moreover, the GenEDA-7B demonstrates further improvements over the GenEDA-1B, indicating that potential gains can be achieved by employing more powerful open-sourced base models.

Natural language specification reasoning case study. We present detailed case studies for these results in Fig. 6. We compare a model-generated response and the corresponding ground truth specification for Task 1, function description generation. The model accurately generates key functionality of the specification, aligning closely with the ground truth. For example, in the functionality section, the model effectively describes how the module handles various control and status registers and memory protection configuration, which matches the ground truth's detailed explanation of register states and operations. These results underline GenEDA's capability to generate high-level natural language descriptions from low-level netlist inputs.

D. Result of RTL Code Reasoning (Task 3)

Baseline solutions. In this task, we choose the advanced commercial LLMs, including GPT-4o and DeepSeek-V3, as the baseline methods. For the open-source LLMs (e.g., DeepSeek-Coder 1B, 7B, and RTLCoder), these models fail to generate high-level RTL and instead produce only gate-level netlist code.

Comparison with baselines. For netlist gate function classification, our encoder achieves a 97% accuracy rate, providing a strong



(a) Effectiveness of alignment (b) Impact of encoder quality

Fig. 9: Ablation study demonstrating the effectiveness of encoder-decoder alignment and the impact of encoder quality.

foundation for our prediction-based alignment paradigm. The impact of encoder quality on alignment performance is further discussed in Section VI-E. TABLE IV evaluates Task 3: Arithmetic RTL Code Reasoning for various models. GenEDA combined with DeepSeek-V3 achieves the highest success rate with a 97% syntax pass rate, 88% GPT Score, and a 60% functional pass rate, outperforming GPT-4o, DeepSeek-V3 alone, and other combinations. Without the prediction alignment, the powerful commercial LLMs alone cannot reason the high-level word-level arithmetics to generate RTL code, and they only generate RTL code using bit-level operations. This demonstrates GenEDA's ability to reversely reconstruct RTL code from netlists with exact functionality.

Notably, even state-of-the-art commercial LLMs like GPT-4o and DeepSeek-V3 achieve less than a 20% functional success rate on our benchmark, highlighting the difficulty of this task. In contrast, existing specification-to-RTL generation benchmarks like RTLLM [39] and VerilogEval [40] report over 60% success with off-the-shelf LLMs like GPT-4, emphasizing the challenge of reverse reconstructing high-level RTL code from low-level bit-blasted netlists.

Arithmetic RTL code reasoning case study. Fig. 8 provides an example of the reasoning process for Task 3. The Chain-of-Thought (CoT) prompt guides the model in two steps: (1) Understanding the arithmetic circuit function: The model reasons about the circuit's gate annotations to identify components like adders, multipliers, and comparators, and determines their combined functionality. These predictions are then annotated back onto the original netlist text, which is provided to the LLM as input for further reasoning. (2) Writing the RTL code: Based on the identified functionality, the model generates RTL code using word-level operations, successfully implementing the circuit's logic. This case study illustrates the effectiveness of GenEDA in generating correct and interpretable RTL code, bridging low-level gate details with high-level functional implementations.

E. Ablation Study

In Fig. 9, we present ablation studies to demonstrate the effectiveness of our encoder-decoder alignment and encoder quality in improving generative reasoning tasks. We conduct experiments by selectively removing the encoder alignment or using less effective encoders to assess their impact on task performance. These studies allow us to isolate and understand the contributions of different components of GenEDA to its overall performance.

Effectiveness of alignment with encoders. GenEDA alignment paradigm 1 is achieved by cross-modally fine-tuning the LLMs using netlist encoder embeddings. In this ablation study, we remove the encoder alignment and only perform supervised fine-tuning (SFT) of the LLMs using the same prompts and labels as in GenEDA. As shown in Fig. 9 (a), removing the encoder alignment significantly decreases model performance on Task 1 and Task 2 across all metrics. Notably, the GPT scores drop sharply from 0.62 to 0.32 on Task 1 and from 0.51 to 0.21 on Task 2, highlighting the effectiveness of embedding alignment.

Impact of encoder quality. GenEDA alignment paradigm 2 heavily relies on the accuracy of gate functionality classification from the encoder. In this ablation study, we replace the high-quality encoder NetTAG [19] with a less effective baseline, GNN-RE [37]. This results in a significant drop in classification accuracy from 97% to 83%. Consequently, the performance of reconstructed RTL code also degrades, with syntax accuracy dropping from 96% to 91% and function accuracy decreasing from 69% to only 40%, as shown in Fig. 9 (b). This demonstrates the importance of using high-quality encoders and their impact on the generation after alignment.

VII. Discussion

A. Extending GenEDA Alignment to Other Circuit Stages and Tasks

Beyond the netlist stage addressed in this work, GenEDA's encoder-decoder alignment framework can be extended to various stages in the circuit design flow. At the RTL stage, even though the same RTL functionality can be generated, different circuit structures can yield significantly varying PPA characteristics. By leveraging structural RTL information captured by RTL encoders, LLMs can potentially enable structure-aware generation, generating more optimized RTL code with better PPA metrics. Additionally, at the layout stage, GenEDA can be adapted to handle cross-modal inputs, such as layout images and netlist graphs. This might enable the direct generation of macro positions on a chip by learning from image representations, thus enabling more efficient and optimized physical design generation.

B. Potential Application of Generative Netlist Functional Reasoning

GenEDA can reason the detailed functionality from netlists, which can significantly benefit verification and optimization processes. Before loading the netlists into EDA tools, GenEDA can guide the selection of appropriate strategies for verifying or optimizing different parts of the design, such as datapaths or control logic. Additionally, it can assist in verifying the equivalence of netlists by transforming them into higher-level RTL code, making the verification process more scalable. In hardware security, GenEDA can be applied to detect malicious hardware trojans. By analyzing netlists, it can identify unexpected or unauthorized functional behaviors, helping to ensure the integrity and security of hardware systems.

VIII. Conclusion and Future Work

In this paper, we present GenEDA, a framework that aligns multimodal circuit encoders and decoders for advanced generative netlist functional reasoning tasks. We align the state-of-the-art netlist encoder with both trainable and frozen LLM decoders through two alignment paradigms. Our experiments and ablation studies demonstrate the effectiveness of this approach, with GenEDA significantly enhancing the performance of state-of-the-art LLMs, showcasing the critical role of integrating both graph and text circuit modalities for complex netlist tasks. Future work will explore extending this alignment framework to other stages of the circuit design flow, such as RTL code generation and layout-stage tasks, further enhancing the capabilities of GenEDA for diverse EDA applications.

IX. Acknowledgement

This research was supported by Hong Kong Research Grants Council (RGC) CRF-YCRG Grant C6003-24Y, GRF 16200724, and ACCESS – AI Chip Center for Emerging Smart Systems, sponsored by the InnoHK initiative of the Innovation and Technology Commission of the Hong Kong Special Administrative Region Government.

REFERENCES

[1] L. Chen, Y. Chen, Z. Chu *et al.*, "Large circuit models: opportunities and challenges," *Science China Information Sciences*, 2024.

[2] W. Fang, J. Wang, Y. Lu, S. Liu, Y. Wu, Y. Ma, and Z. Xie, "A survey of circuit foundation model: Foundation ai models for vlsi circuit design and eda," *arXiv preprint arXiv:2504.03711*, 2025.

[3] W. Fang, S. Liu, H. Zhang, and Z. Xie, "A self-supervised, pre-trained, and cross-stage-aligned circuit encoder provides a foundation for various design tasks," in *ASP-DAC*, 2025.

[4] Z. Shi, Z. Zheng, S. Khan, J. Zhong, M. Li, and Q. Xu, "Deepgate3: towards scalable circuit representation learning," *arXiv preprint arXiv:2407.11095*, 2024.

[5] Z. Shi, H. Pan, S. Khan, M. Li, Y. Liu, J. Huang, H.-L. Zhen, M. Yuan, Z. Chu, and Q. Xu, "DeepGate2: Functionality-aware circuit representation learning," in *ICCAD*, 2023.

[6] M. Li, S. Khan, Z. Shi, N. Wang, H. Yu, and Q. Xu, "DeepGate: Learning neural representations of logic gates," in *DAC*, 2022.

[7] Z. Wang, C. Bai, Z. He, G. Zhang, Q. Xu, T.-Y. Ho, B. Yu, and Y. Huang, "Functionality matters in netlist representation learning," in *Design Automation Conference (DAC)*, 2022.

[8] W. Fang, S. Liu, J. Wang, and Z. Xie, "Circuitfusion: multimodal circuit representation learning for agile chip design," in *International Conference on Learning Representations (ICLR)*, 2025.

[9] H. Wu, H. Zheng, Y. Pu, and B. Yu, "Circuit representationlearning with masked gatemodeling and verilog-aigalignment," in *International Conference on Learning Representations (ICLR)*, 2025.

[10] C. Deng, Z. Yue *et al.*, "Less is more: Hop-wise graph attention for scalable and generalizable learning on circuits," in *DAC*, 2024.

[11] C. Xu, P. Sharma, T. Wang, and L. W. Wills, "Fast, robust and transferable prediction for hardware logic synthesis," in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2023.

[12] Y. Zhang, Z. Yu *et al.*, "Mg-verilog: Multi-grained dataset towards enhanced llm-assisted verilog generation," *arXiv preprint arXiv:2407.01910*, 2024.

[13] Z. Pei, H.-L. Zhen, M. Yuan, Y. Huang, and B. Yu, "Betterv: Controlled verilog generation with discriminative guidance," *arXiv preprint arXiv:2402.03375*, 2024.

[14] 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," *IEEE LLM-Aided Design (LAD)*, 2023.

[15] Z. Yan, W. Fang, M. Li, M. Li, Z. Yan, S. Liu, Z. Xie, and H. Zhang, "AssertLLM: Generating and evaluating hardware verification assertions from design specifications via multi-LLMs," in *ASP-DAC*, 2025.

[16] R. Kande, H. Pearce *et al.*, "(security) assertions by large language models," *IEEE Transactions on Information Forensics and Security (TIFS)*, 2024.

[17] 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 *Workshop on Machine Learning for CAD (MLCAD)*, 2023.

[18] Z. Zheng, S. Huang, J. Zhong *et al.*, "Deepgate4: Efficient and effective representation learning for circuit design at scale," in *International Conference on Learning Representations (ICLR)*, 2025.

[19] W. Fang, W. Li, S. Liu, Y. Lu, H. Zhang, and Z. Xie, "Nettag: A multimodal rtl-and-layout-aligned netlist foundation model via text-attributed graph," in *Design Automation Conference (DAC)*, 2025.

[20] S. Liu, W. Fang, Y. Lu, Q. Zhang, H. Zhang, and Z. Xie, "Rtlcoder: Fully open-source and efficient llm-assisted rtl code generation technique," *IEEE TCAD*, 2024.

[21] X. Yao, H. Li, T. H. Chan, W. Xiao, M. Yuan, Y. Huang, L. Chen, and B. Yu, "Hdldebugger: Streamlining hdl debugging with large language models," *arXiv preprint arXiv:2403.11671*, 2024.

[22] Y. Liu, C. Xu, Y. Zhou, Z. Li, and Q. Xu, "DeepRTL: Bridging verilog understanding and generation with a unified representation model," in *International Conference on Learning Representations (ICLR)*, 2025.

[23] N. Wu, Y. Li, C. Hao, S. Dai, C. Yu, and Y. Xie, "Gamora: Graph learning based symbolic reasoning for large-scale boolean networks," in *ACM/IEEE Design Automation Conference (DAC)*, 2023.

[24] Z. Wang, C. Bai, Z. He, G. Zhang, Q. Xu, T.-Y. Ho, Y. Huang, and B. Yu, "Fgnn2: A powerful pre-training framework for learning the logic functionality of circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2024.

[25] A. Kuehlmann, V. Paruthi, F. Krohm, and M. K. Ganai, "Robust boolean reasoning for equivalence checking and functional property verification," *IEEE TCAD*, 2002.

[26] P. Subramanyan, N. Tsiskaridze, K. Pasricha, D. Reisman, A. Susnea, and S. Malik, "Reverse engineering digital circuits using functional analysis," in *DATE*, 2013.

[27] S. Khan, Z. Shi, Z. Zheng, M. Li, and Q. Xu, "Deepseq2: Enhanced sequential circuit learning with disentangled representations," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2025.

[28] J. Liu, J. Zhai, M. Zhao, Z. Lin, B. Yu, and C. Shi, "Polargate: Breaking the functionality representation bottleneck of and-inverter graph neural network," in *ICCAD*, 2024.

[29] M. Liu, T.-D. Ene, R. Kirby, C. Cheng, N. Pinckney *et al.*, "ChipNeMo: Domain-Adapted LLMs for Chip Design," *arXiv preprint arXiv:2311.00176*, 2023.

[30] K. Chang, Y. Wang, H. Ren, M. Wang, S. Liang, Y. Han, H. Li, and X. Li, "Chipgpt: How far are we from natural language hardware design," *arXiv preprint arXiv:2305.14019*, 2023.

[31] C. Sun, C. Hahn, and C. Trippel, "Towards improving verification productivity with circuit-aware translation of natural language to systemverilog assertions," in *International Workshop on Deep Learning-aided Verification (DAV)*, 2023.

[32] X. Yao, Y. Wang, X. Li, Y. Lian, R. Chen, L. Chen, M. Yuan, H. Xu, and B. Yu, "Rtlrewriter: Methodologies for large models aided rtl code optimization," *arXiv preprint arXiv:2409.11414*, 2024.

[33] Y. Jiang, X. Lu, Q. Jin, Q. Sun, H. Wu, and C. Zhuo, "Fabgpt: An efficient large multimodal model for complex wafer defect knowledge queries," *arXiv preprint arXiv:2407.10810*, 2024.

[34] H. Wu, Z. He *et al.*, "Chateda: A large language model powered autonomous agent for eda," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[35] W. Li, Z. Wasson, and S. A. Seshia, "Reverse engineering circuits using behavioral pattern mining," in *2012 IEEE international symposium on hardware-oriented security and trust*. IEEE, 2012, pp. 83–88.

[36] A. Gascón, P. Subramanyan, B. Dutertre, A. Tiwari, D. Jovanović, and S. Malik, "Template-based circuit understanding," in *Formal Methods in Computer-Aided Design (FMCAD)*. IEEE, 2014, pp. 83–90.

[37] L. Alrahis, A. Sengupta *et al.*, "Gnn-re: Graph neural networks for reverse engineering of gate-level netlists," *IEEE TCAD*, 2021.

[38] S. D. Chowdhury, K. Yang, and P. Nuzzo, "Reignn: State register identification using graph neural networks for circuit reverse engineering," in *ICCAD*, 2021.

[39] Y. Lu, S. Liu, Q. Zhang, and Z. Xie, "RTLLM: An open-source benchmark for design rtl generation with large language model," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2024.

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

[41] F. Corno, M. S. Reorda, and G. Squillero, "RT-level ITC'99 benchmarks and first ATPG results," *IEEE Design & Test of Computers*, 2000.

[42] *OpenCores: The reference community for Free and Open Source gateware IP cores*, https://opencores.org/.

[43] A. Amid, D. Biancolin, A. Gonzalez, D. Grubb, S. Karandikar, H. Liew, A. Magyar, H. Mao, A. Ou, N. Pemberton *et al.*, "Chipyard: Integrated design, simulation, and implementation framework for custom SoCs," *IEEE Micro*, 2020.

[44] VexRiscv, "VexRiscv: A FPGA friendly 32 bit RISC-V CPU implementation," 2022. [Online]. Available: https://github.com/SpinalHDL/VexRiscv

[45] C. Wolf, J. Glaser, and J. Kepler, "Yosys-a free verilog synthesis suite," in *Austrian Workshop on Microelectronics (Austrochip)*, 2013.

[46] D. Guo, Q. Zhu, D. Yang, Z. Xie, K. Dong, W. Zhang, G. Chen, X. Bi, Y. Wu, Y. Li *et al.*, "Deepseek-coder: When the large language model meets programming–the rise of code intelligence," *arXiv preprint arXiv:2401.14196*, 2024.

[47] A. Liu, B. Feng, B. Xue, B. Wang, B. Wu, C. Lu, C. Zhao, C. Deng, C. Zhang, C. Ruan *et al.*, "Deepseek-v3 technical report," *arXiv preprint arXiv:2412.19437*, 2024.

[48] H. Liu, C. Li, Q. Wu, and Y. J. Lee, "Visual instruction tuning," *Advances in neural information processing systems*, vol. 36, 2024.

[49] J. Li, D. Li, S. Savarese, and S. Hoi, "Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models," in *ICML*, 2023.

[50] J. Li, D. Li, C. Xiong, and S. Hoi, "Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation," in *International Conference on Machine Learning (ICML)*, 2022.