Bridging Circuit Graphs and Language: A Cross-Modal Transformer for Netlist-to-Text Alignment and Generation

Anonymous Author(s)

Abstract

As integrated circuit (IC) design grows in complexity, foundation models for circuits have emerged in two forms: encoders for predictive tasks and decoders for generative tasks. While LLM-based circuit decoders excel at producing circuit-related text, their reliance on a text-based latent space limits their ability to capture structural information from the circuit-graph latent space. Recent work bridges this gap using an MLP-based connector trained with autoregressive loss, but struggles to align complex netlists with linguistically contextualized text and tends to prioritize next-token prediction over summarizing critical netlist information. We propose NetlistFormer, a cross-modal circuit transformer for netlist-to-text alignment and generation, which uses learnable query vectors over the netlist graph in a two-stage framework. In the first stage, the queries act as functional sub-structure selectors that extract and compress key structural information from the netlist graph under textual guidance, via a multi-objective cross-modal alignment beyond autoregressive loss. In the second stage, the resulting queries are passed to an LLM for netlist-language generation. Experiments show that NetlistFormer consistently outperforms general-purpose LLMs and MLP-based connector models across standard similarity metrics and domain-aware keyword-level metrics. To our knowledge, NetlistFormer is the first circuit generative transformer trained with multi-objective cross-modal alignment. Our source code is available at https://anonymous.4open.science/r/NetlistFormer-E5C0.

Keywords

Graph Representation Learning, Graph-Language Alignment, Cross-Modal Learning, Large Language Model

ACM Reference Format:

Anonymous Author(s). 2026. Bridging Circuit Graphs and Language: A Cross-Modal Transformer for Netlist-to-Text Alignment and Generation. In *Proceedings of Proceedings of the 32nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '26).* ACM, New York, NY, USA, 9 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

As the complexity and cost of integrated circuit (IC) design continue to grow, conventional electronic design automation (EDA) techniques are increasingly revealing their limitations. Accordingly, researchers are actively exploring new paradigms for IC design, such as AI-based EDA. More recently, the adaptation of foundation AI models to IC design has emerged as an active research direction, leading to the development of circuit foundation models [5, 15] that are expected to provide more general and scalable AI solutions for IC design.

Existing circuit foundation models fall into two broad categories: circuit encoders for predictive tasks and circuit decoders for generative tasks. Circuit encoders [12, 13, 14] map circuits into general-purpose embeddings that can be fine-tuned for various downstream EDA tasks such as circuit functionality reasoning [14, 29, 30, 23, 34] and circuit quality prediction [13, 36, 8, 38]. In contrast, circuit decoders are typically built on Large Language Models (LLMs) and generate circuit-related text such as functional descriptions, implementation details, and verification assertions [11, 19]. While effective at producing fluent text, circuit decoders operate within a textual latent space and thus cannot fully capture the structural information of circuits residing in the circuit-graph latent space.

To bridge this gap, GenEDA [10] introduces a modality connector that transforms circuit embeddings from an encoder into textual embeddings compatible with the LLM decoder, jointly training the modality connector and the decoder with an autoregressive loss. However, this approach has two key limitations. First, the connector is realized as a simple MLP, whose limited representational capacity makes it difficult to align the complex structural information of netlists with linguistically contextualized text. Second, training relies solely on an autoregressive loss, which biases the model toward next-token prediction rather than encouraging it to explicitly summarize the critical structural information embedded in the netlist.

To address these limitations, we propose **NetlistFormer**, a Cross-Modal Circuit Transformer for Netlist-to-Text Alignment and Generation that consists of two query-based stages. To address the first limitation, NetlistFormer replaces the static MLP with a Transformer-based connector equipped with a set of learnable query vectors that extract structural features from a fixed netlist encoder. Rather than uniformly aggregating information from the netlist, these queries selectively attend to the parts most relevant to the target description, thereby enabling fine-grained alignment between complex netlist structures and linguistically contextualized text. To address the second limitation, NetlistFormer goes beyond autoregressive supervision by pursuing three complementary objectives that align netlist and text representations at the global, token, and generative levels, jointly encouraging the queries to capture essential netlist information for describing the design. This explicit alignment pressure pushes the queries to summarize the netlist information that is most informative for describing the design, rather than merely the information that happens to help predict the next token. Building on these two design choices, NetlistFormer is trained in two stages: (1) *netlist-language representation learning*, in which the queries

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

KDD '26, Jeju, South Korea

© 2026 Copyright held by the owner/author(s). Publication rights licensed to ACM.

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX

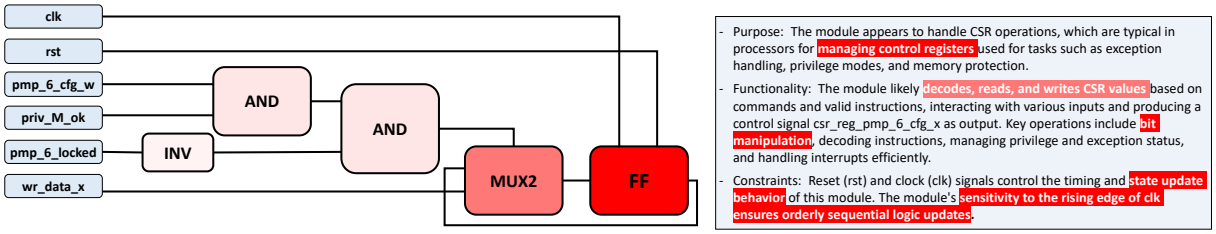

Figure 1: Visualization of NetlistFormer's attention over circuit nodes for text generation. Each node in the netlist is colored according to the attention weight it receives from the learnable queries in the first stage, where darker colors indicate stronger attention. NetlistFormer focuses on the nodes whose function and connectivity determine the design intent expressed in the target text, indicating that the queries learn to extract the circuit information that is essential for text generation.

learn under textual guidance which structural information to extract from the netlist, and (2) *netlist-language generative learning*, where the learned queries go through an LLM to generate a natural-language response to an input prompt. As illustrated in Figure 1, the resulting queries concentrate on the circuit nodes whose function and connectivity shape the design intent expressed in the text. This enables the model to establish more sophisticated logical bridges between the physical connectivity of the circuit and the output text.

We conducted extensive experiments to evaluate the effectiveness of NetlistFormer in netlist-to-text alignment and generation. Our results show that NetlistFormer consistently outperforms a range of strong baselines, including general-purpose LLMs and the MLP-based connector model, across both standard similarity metrics and domain-aware keyword-level metrics. Moreover, we conducted a qualitative analysis that visualizes the generated specifications alongside the corresponding ground-truth descriptions, suggesting that NetlistFormer is able to faithfully capture key aspects of each design such as its interface, purpose, and core functionality. Overall, these results demonstrate that NetlistFormer significantly improves the alignment between netlist structure and natural language, which in turn leads to higher-quality specifications in downstream circuit-to-text generation tasks. To the best of our knowledge, NetlistFormer is the first circuit generative transformer trained with multi-objective cross-modal alignment.

2 Related Works

2.1 Circuit Foundation Model

The pretrain–finetune paradigm has recently driven a wave of circuit foundation models that aim to provide generalizable AI building blocks for EDA. These models can be broadly divided into two categories: encoder-based predictors and decoder-based generators. Encoder-based models pretrain structural representations of circuits and reuse them for predictive tasks such as reverse functional reasoning and early-stage design-quality estimation. Many studies [23, 30, 29, 20, 41, 34] operate on AIG netlists and adopt graph-based learning to capture the underlying topology. A few recent works [13, 36, 12] additionally incorporate textual or visual modalities, yet such fusion is confined to the input side of the encoder and is never aligned with an LLM, which would be required to unlock generative capability. Decoder-based models leverage LLMs as the backbone of forward-generation pipelines for tasks such as assertion synthesis [25, 4, 11, 32], debugging [39], and optimization [28, 40]. Because RTL itself carries rich semantics that are largely accessible to language models, even a text-only T5 backbone has proven competitive on these tasks. When other modalities are introduced, prior work [40, 11] has mainly relied on circuit images, an input type that existing multimodal LLMs already handle, while leaving the more informative graph-structured netlist representation largely untouched on the decoder side.

GenEDA [10] addresses this gap by proposing the first cross-modal encoder–decoder alignment framework that maps circuit structural features and LLM textual representations into a shared latent space. NetlistFormer extends this architectural direction by replacing the lightweight MLP connector of GenEDA with a query-based two-stage transformer trained jointly with contrastive, matching, and netlist-grounded generation objectives, providing a more expressive interface between circuit structure and language.

2.2 Functional Reasoning on Netlist

Understanding what a gate-level netlist actually does at a high level is important for verification, logic optimization, datapath synthesis, and hardware security, especially when the original specifications are no longer available. Prior work in this direction generally falls into two categories: formal methods that analyze a circuit using symbolic rules [21, 24, 31, 16], and learning-based methods that classify the function of each gate from data [2, 6, 34, 37, 8].

Formal methods [21, 24, 31, 16] typically break a netlist into smaller subcircuits and try to match each piece with a known component in a reference library through exhaustive verification. This works well when a circuit closely follows a cataloged component, but the process becomes slow as circuits grow larger, depends heavily on how complete the library is, and often misses circuits that implement the same function in a slightly different form. Learning-based methods [2, 6, 34, 37, 8] take a different route: they train graph neural networks (GNNs) to predict the role of each gate from its surrounding connections in the netlist. These models recognize familiar components fairly well, but they only assign labels at the level of individual gates, so they struggle to handle unfamiliar functions or to describe the behavior of an entire circuit.

GenEDA [10] shifts netlist functional reasoning from per-gate classification to a generation problem, producing natural-language specifications directly from a low-level netlist. NetlistFormer follows this direction and further improves the fidelity of the resulting

descriptions by aligning structural cues with language under multi-objective supervision, allowing the model to capture the role and behavior of the entire circuit more faithfully.

3 Preliminary

In a typical digital design flow, a register-transfer level (RTL) description is synthesized into a gate-level netlist composed of standard cells and nets. A gate-level netlist describes a digital circuit as a collection of logic cells and their electrical connections. Each cell implements a Boolean function, such as an inverter, NAND gate, or NOR gate, and contains input and output pins through which signals enter and leave the cell. A net connects one driver pin to one or more sink pins, thereby defining the signal propagation between cells. We represent a gate-level netlist as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ denotes the set of nodes corresponding to cells, and $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ denotes the set of edges representing signal flow in the netlist. For a given netlist $\mathcal{G}$, a pretrained netlist encoder produces a node embedding $\mathbf{h}_v \in \mathbb{R}^{d_g}$ for each node $v \in \mathcal{V}$, forming the node embedding matrix $H \in \mathbb{R}^{|\mathcal{V}| \times d_g}$. In addition, a ground-truth natural-language description $T = (t_1, \ldots, t_L)$ for the netlist is provided, where each token $t_l \in \mathcal{W}$ belongs to a predefined vocabulary $\mathcal{W}$, a special token [CLS] is prepended as the first token to aggregate the sentence-level representation, and L is the number of tokens in the description T. Given a dataset $\mathcal{D} = \{(\mathcal{G}_i, T_i)\}_{i=1}^{N}$, the goal of this work is to learn the conditional distribution $p_\theta(T \mid \mathcal{G})$ for generating a text description from the netlist graph.

4 Methodology

We propose **NetlistFormer**, a generative netlist foundation model that learns structural summaries for language generation through a query-based two-stage transformer framework. To effectively bridge the gap between circuit-graph and language modalities, we train the model in two stages: (1) *netlist-language representation learning*, in which the queries learn under textual guidance which structural information to extract from the netlist, and (2) *netlist-language generative learning*, where the learned queries go through an LLM to generate a natural-language response to an input prompt.

4.1 Stage 1: Netlist-Language Representation Learning

In this stage, we train a set of K randomly initialized but learnable query tokens $Q \in \mathbb{R}^{K \times d_q}$ so that, at inference time, passing them together with an input netlist through the backbone yields query representations that capture key structural information for downstream text generation. Textual descriptions are only used during training, where they provide the supervision signal that guides the queries toward language-relevant structural features. To this end, we jointly optimize three objectives applied to a shared transformer backbone described below.

4.1.1 Backbone Architecture. Building on Q-Former of BLIP-2 [22], we instantiate the backbone model as a single transformer [33]. It is worth noting that although we retain Q-Former's three-mode design and shared self-/cross-attention scheme, the role of the learnable queries differs fundamentally from the vision-language setting: rather than soft region selectors over a fixed-size visual patch grid, our queries act as functional sub-structure selectors over a variable-size circuit graph, attending to the cells whose connectivity and Boolean function jointly determine the design intent (cf. Figure 1). This shift in the queries' semantic role drives the alignment objectives introduced in Sections 4.1.2–4.1.4.

As illustrated in Figure 2, the model operates with three modes depending on the input configuration: (a) a *Query Transformer* f_q that processes the query tokens Q, (b) a *Text Transformer* f_t that processes the text tokens T, and (c) a *Fusion Transformer* f_m in which Q and T are jointly processed. Each transformer block consists of an input projection, a self-attention layer, a cross-attention layer, and a feed-forward network (FFN). The two attention layers share their weights across all modes, while weights of the input projection and the FFN are kept modality-specific, meaning that either the query-side or the text-side parameters are used depending on the current input token's type. The cross-attention layer is invoked only when there is external information to attend to, where the specific attention sources for each loss term are described in the corresponding subsections.

4.1.2 Netlist–Text Contrastive Loss $\mathcal{L}_{NTC}$. The contrastive objective $\mathcal{L}_{\text{NTC}}$ encourages the queries to extract the global structural information from the netlist needed to capture the overall meaning of the textual description. To this end, we encode each modality through a separate pathway and align the resulting representations in a shared embedding space. The corresponding architecture is shown in Figure 2a.

Separate Encoding. For a given mini-batch of size B, the Query Transformer f_q takes the learnable query tokens Q as input and produces netlist representations by attending to the node embedding H obtained from the frozen netlist encoder through cross-attention. Meanwhile, the Text Transformer f_t processes the text description T without external reference and produces the token-wise textual representations. That is,

$$Z^g = f_q(Q, H) \in \mathbb{R}^{B \times K \times d}, \quad Z^t = f_t(T) \in \mathbb{R}^{B \times L \times d}. \tag{1}$$

where the output $\mathbf{z}^t = Z^t_{[\text{CLS}]} \in \mathbb{R}^d$ for the [CLS] token serves as the global representation of the entire description, and B is the batch size. Note that cross-attention is not invoked since no external information is provided.

Contrastive Loss. Given the representations Z^g and $\mathbf{z}^t$, the contrastive loss aligns each netlist representation with its paired description by contrasting their similarity against the similarities to the other descriptions in the same mini-batch, and symmetrically for the reverse direction. Let $\mathbf{z}^g_{i,k} \in \mathbb{R}^d$ denote the k-th query representation of the i-th netlist $\mathcal{G}_i$ in the mini-batch, and let $\mathbf{z}^t_i \in \mathbb{R}^d$ denote the [CLS] representation of the i-th netlist description T_i. Since each netlist has K query representations, we define the netlist-to-text similarity for each $(\mathcal{G}_i, T_j)$ pair in the mini-batch as the average cosine similarity over all K queries, and analogously for the reverse direction:

$$s_{g \to t}(i, j) = \frac{1}{K} \sum_{k=1}^{K} \text{sim}\left(\mathbf{z}^g_{i,k}, \mathbf{z}^t_j\right), \quad s_{t \to g}(i, j) = \frac{1}{K} \sum_{k=1}^{K} \text{sim}\left(\mathbf{z}^t_i, \mathbf{z}^g_{j,k}\right). \tag{2}$$

where $\text{sim}(\cdot)$ denotes cosine similarity. Treating the diagonal pairs (i, i) as positives and the off-diagonal pairs as in-batch negatives,

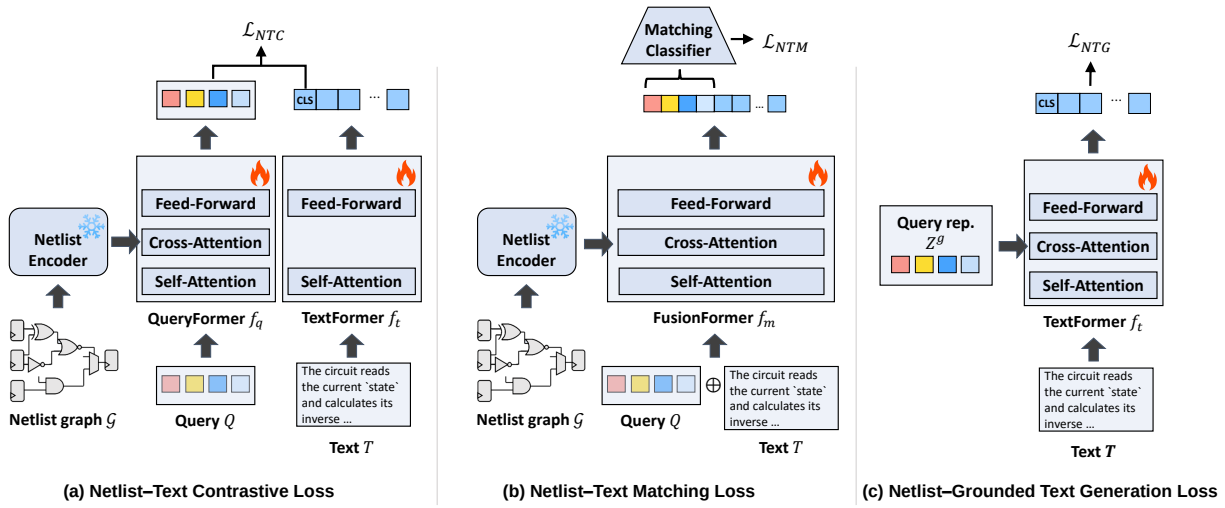

Figure 2: The three loss components that constitute the first stage of NetlistFormer.

the bidirectional contrastive loss is given by:

$$\mathcal{L}_{\text{NTC}} = -\frac{1}{2B}\sum_{i=1}^{B}\left[\log\frac{\exp\big(s_{g\to t}(i,i)\big)}{\sum_{j=1}^{B}\exp\big(s_{g\to t}(i,j)\big)} + \log\frac{\exp\big(s_{t\to g}(i,i)\big)}{\sum_{j=1}^{B}\exp\big(s_{t\to g}(i,j)\big)}\right]. \tag{3}$$

4.1.3 Netlist–Text Matching Loss $\mathcal{L}_{NTM}$. While $\mathcal{L}_{\text{NTC}}$ aligns the netlist and text representations at a global level through separate encoding pathways, it does not directly model token-level interactions between the two modalities. We introduce a matching objective $\mathcal{L}_{\text{NTM}}$ that complements this by jointly encoding queries and text tokens in a shared self-attention space, so that each query can form fine-grained associations with text tokens. See Figure 2b illustrating the corresponding architecture.

Hard Negative Sampling. Since performing joint encoding for every $(\mathcal{G}_i, T_j)$ pair is impractical, we restrict each anchor to a small set of negatives. To ensure that these negatives provide an informative learning signal, we sample *hard negatives* with a high similarity to the anchor. Concretely, for each sample i, we draw a negative-netlist index j_i^{g-} and a negative-text index j_i^{t-} as follows:

$$j_i^{g-} \sim \text{softmax}_{j\neq i}\big(s_{t\to g}(i,j)\big), \quad j_i^{t-} \sim \text{softmax}_{j\neq i}\big(s_{g\to t}(i,j)\big), \tag{4}$$

so that samples with higher similarity are more likely to be selected. This yields three pairs per sample: a positive pair $(\mathcal{G}_i, T_i)$, a negative-netlist pair $(\mathcal{G}_{j_i^{g-}}, T_i)$, and a negative-text pair $(\mathcal{G}_i, T_{j_i^{t-}})$.

Joint Encoding. Each pair $(\mathcal{G}, T)$ is processed by the Fusion Transformer f_m, in which queries Q and text tokens T are concatenated to directly interact through the shared self-attention layers, while the node embeddings H are injected through cross-attention. As a result, for each sample i, we obtain:

$$Z_i^{+} = f_m(Q\oplus T_i, H_i),\ Z_i^{g-} = f_m(Q\oplus T_i, H_{j_i^{g-}}),\ Z_i^{t-} = f_m(Q\oplus T_{j_i^{t-}}, H_i) \in \mathbb{R}^{(K+L)\times d}, \tag{5}$$

each consisting of $K+L$ fused representations corresponding to respective input tokens. We extract the K query positions and map their averages to matching logits by a classifier W_{NTM}:

$$\hat{\ell}_i^{+} = W_{\text{NTM}}\left(\frac{1}{K}\sum_{k=1}^{K}\mathbf{z}_{i,k}^{+}\right),\quad \hat{\ell}_i^{g-} = W_{\text{NTM}}\left(\frac{1}{K}\sum_{k=1}^{K}\mathbf{z}_{i,k}^{g-}\right),\quad \hat{\ell}_i^{t-} = W_{\text{NTM}}\left(\frac{1}{K}\sum_{k=1}^{K}\mathbf{z}_{i,k}^{t-}\right), \tag{6}$$

where $\mathbf{z}_{i,k}^{+}, \mathbf{z}_{i,k}^{g-}, \mathbf{z}_{i,k}^{t-} \in \mathbb{R}^d$ denote the k-th query representations for the three pairs, respectively.

Matching Loss. Treating each pair as an independent binary decision with the positive pair labeled as 1 and both negative pairs as 0, $\mathcal{L}_{\text{NTM}}$ is given by the average binary cross-entropy as follows:

$$\mathcal{L}_{\text{NTM}} = \frac{1}{3B}\sum_{i=1}^{B}\Big[\text{CE}(\hat{\ell}_i^{+}, 1) + \text{CE}(\hat{\ell}_i^{g-}, 0) + \text{CE}(\hat{\ell}_i^{t-}, 0)\Big]. \tag{7}$$

4.1.4 Netlist-Grounded Text Generation Loss $\mathcal{L}_{NTG}$. Together, $\mathcal{L}_{\text{NTC}}$ and $\mathcal{L}_{\text{NTM}}$ align netlist and text representations at the global and token level, respectively. However, both objectives are discriminative in nature: they only require the queries to distinguish matched netlist–text pairs from mismatched ones, and do not directly enforce that the queries retain enough netlist information to actually *produce* the corresponding description. Since the ultimate role of the queries in Stage 2 is to drive language generation through an LLM, this discriminative supervision alone is not sufficient. To close this gap, we introduce a generation objective $\mathcal{L}_{\text{NTG}}$ that trains the query representations to reconstruct the textual description from the netlist, thereby ensuring that the queries preserve the structural information required for downstream text generation. The architecture corresponding to this objective is illustrated in Figure 2c.

Generation from Query Representations. Rather than re-encoding the netlist via a separate pathway, we reuse the query representations Z^g produced from Eq. 1 as a key-value cache. The Text Transformer f_t then takes the description T_i for each sample i as input under teacher forcing, and autoregressively predicts the next token under a causal mask. Within f_t, the text tokens look back at preceding text tokens through self-attention and reference the cached Z_i^g through cross-attention:

$$\hat{T}_i = (\hat{t}_{i,1}, \ldots, \hat{t}_{i,L_i}) = f_t(T_i, Z_i^g) \in \mathbb{R}^{L_i\times d}, \tag{8}$$

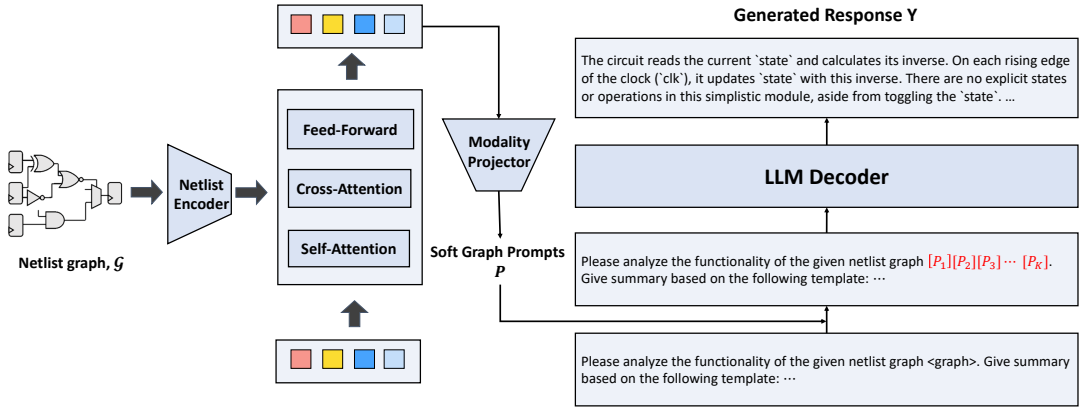

Figure 3: The architecture for Stage 2 of NetlistFormer Architecture.

where $\hat{t}_{i,l}$ denotes the hidden state at position l and L_i is the number of tokens in the i-th description.

Generation Loss. A language modeling head is attached on top of $\hat{T}_i$ to produce the next-token distribution p_θ. The generation loss is the standard autoregressive negative log-likelihood as follows:

$$\mathcal{L}_{\text{NTG}} = -\frac{1}{B}\sum_{i=1}^{B}\frac{1}{L_i}\sum_{l=1}^{L_i}\log p_\theta\left(t_{i,l} \mid t_{i,<l}, Z_i^g\right). \tag{9}$$

4.2 Stage 2: Circuit-to-Language Generative Learning

In Stage 2, we fine-tune a pretrained LLM to generate natural-language responses according to the input prompt related to a given netlist, using the query representations obtained from the Query Transformer f_q trained in Stage 1. However, these representations are not directly aligned with the token representation space of the LLM, and are not yet specialized for natural-language response generation. To bridge this gap, in Stage 2, we introduce a trainable modality projector W_{LLM} between the output queries from Stage 1 and the LLM and perform supervised fine-tuning on natural-language generation tasks such as circuit summarization, as illustrated in Figure 3. Throughout this process, the LLM backbone is kept frozen, and only the low-rank adapters (LoRA) [18] and the modality projector are trained, which preserves the original language generation capability while efficiently strengthening circuit-language alignment.

4.2.1 Multimodal Input Construction. The input for the i-th sample consists of the query representations Z_i^g, a natural-language prompt token sequence $X_i = (x_{i,1}, \ldots, x_{i,M})$, and a target response sequence $Y_i = (y_{i,1}, \ldots, y_{i,R_i})$. The prompt sequence X_i contains a special placeholder token <graph> at position $m^\star$, which marks where the netlist information is to be inserted as a continuous prompt.

To produce this continuous prompt, we first apply f_q to extract K query representations Z_i^g and project them into the latent token space of the LLM through the linear projector $W_{\text{LLM}} \in \mathbb{R}^{d\times d'}$:

$$Z_i^g = f_q(Q, H_i) \in \mathbb{R}^{K\times d}, \quad P_i = Z_i^g W_{\text{LLM}} \in \mathbb{R}^{K\times d'}, \tag{10}$$

where $P_i = (\mathbf{p}_{i,1}, \ldots, \mathbf{p}_{i,K})$ denotes the resulting *soft graph prompt* that is directly compatible with the input space of LLM. Then, the final multimodal input sequence is constructed by replacing the <graph> token with the soft graph prompt as follows:

$$\tilde{X}_i = \left(f_{\text{emb}}(x_{i,1}), \ldots, f_{\text{emb}}(x_{i,m^\star-1}), \mathbf{p}_{i,1}, \ldots, \mathbf{p}_{i,K}, f_{\text{emb}}(x_{i,m^\star+1}), \ldots, f_{\text{emb}}(x_{i,M})\right), \tag{11}$$

where f_{emb} denotes the token embedding function of the LLM. This formulation enables the LLM to process the netlist and the text prompt as a unified token sequence in a continuous representation space, rather than as two separate modalities.

4.2.2 Response Generation Loss $\mathcal{L}_{RG}$. In Stage 2, the LLM is trained to autoregressively generate the target response Y_i conditioned on the multimodal input $\tilde{X}_i$. We optimize the model by minimizing the negative log-likelihood of Y_i:

$$\mathcal{L}_{\text{RG}} = -\frac{1}{B}\sum_{i=1}^{B}\frac{1}{R_i}\sum_{r=1}^{R_i}\log p_\phi\left(y_{i,r} \mid y_{i,<r}, \tilde{X}_i\right), \tag{12}$$

where p_ϕ denotes the next-token distribution conditioned on the multimodal input prompt and the previously generated response tokens and ϕ collects the trainable parameters of W_{LLM} and LoRA.

5 Experiments

5.1 Experimental Setup

Tasks. In our experiments, we evaluate two tasks, both of which require transforming a gate-level netlist into a high-level natural-language description of the underlying design. As described in Section 3, the gate-level netlist is synthesized from the corresponding register-transfer level (RTL) design and serves as the input to the evaluated models. Following [10], the first task, Function Description Generation (Task 1), captures the overall behavior of the circuit by asking the model to characterize the design along four axes—interface, purpose, functionality, and constraints—given only its netlist. The second task, Implementation Detail Generation (Task 2), focuses on how the design realizes its function and prompts the model to articulate the combinational logic, sequential behavior, and control flow.

Datasets. Our circuit dataset is built from a range of publicly available RTL code benchmarks, including ITC99 [7], OpenCores [1], Chipyard [3], and VexRiscv [27]. Following the methodology proposed in [12], we partition large-scale circuits into smaller subcircuit units. Each RTL design is then synthesized into a gate-level netlist with Yosys [35]. The resulting gate-level netlists are then

Table 1: Specification Reasoning results on Task 1 (Functional Description) and Task 2 (Implementation Detail)

Task 1: Functional Description						
Model	BLEU	ROUGE-1	ROUGE-2	ROUGE-L	Emb. Sim.	GPT Score
GPT4o	$4_{\pm0.7}$	$30_{\pm1.4}$	$8_{\pm0.8}$	$16_{\pm1.1}$	$0.83_{\pm0.010}$	$0.20_{\pm0.03}$
DeepSeek-V3	$5_{\pm0.8}$	$32_{\pm1.5}$	$10_{\pm0.9}$	$18_{\pm1.2}$	$0.82_{\pm0.012}$	$0.20_{\pm0.03}$
DeepSeek-1B	$6_{\pm1.0}$	$28_{\pm1.8}$	$10_{\pm1.1}$	$19_{\pm1.4}$	$0.84_{\pm0.015}$	$0.04_{\pm0.03}$
DeepSeek-7B	$5_{\pm1.0}$	$29_{\pm1.8}$	$9_{\pm1.0}$	$19_{\pm1.4}$	$0.82_{\pm0.017}$	$0.18_{\pm0.04}$
GenEDA-1B	$7_{\pm1.5}$	$34_{\pm2.5}$	$11_{\pm1.6}$	$19_{\pm1.9}$	$0.87_{\pm0.022}$	$0.49_{\pm0.05}$
GenEDA-7B	$11_{\pm1.7}$	$41_{\pm2.7}$	$16_{\pm1.8}$	$24_{\pm2.0}$	$0.90_{\pm0.024}$	$\mathbf{0.50}_{\pm0.05}$
NetlistFormer-1B	$\mathbf{24}_{\pm2.0}$	$\mathbf{57}_{\pm2.9}$	$\mathbf{26}_{\pm2.1}$	$\mathbf{35}_{\pm2.4}$	$\mathbf{0.92}_{\pm0.022}$	$0.48_{\pm0.05}$
NetlistFormer-7B	$\mathbf{26}_{\pm2.1}$	$\mathbf{58}_{\pm3.0}$	$\mathbf{28}_{\pm2.2}$	$\mathbf{36}_{\pm2.5}$	$\mathbf{0.93}_{\pm0.020}$	$\mathbf{0.50}_{\pm0.04}$

Task 2: Implementation Detail						
Model	BLEU	ROUGE-1	ROUGE-2	ROUGE-L	Emb. Sim.	GPT Score
GPT4o	$5_{\pm0.4}$	$35_{\pm0.8}$	$10_{\pm0.5}$	$17_{\pm0.6}$	$0.83_{\pm0.006}$	$0.27_{\pm0.02}$
DeepSeek-V3	$4_{\pm0.4}$	$34_{\pm0.8}$	$10_{\pm0.5}$	$17_{\pm0.6}$	$0.81_{\pm0.007}$	$0.21_{\pm0.02}$
DeepSeek-1B	$6_{\pm0.5}$	$39_{\pm1.0}$	$11_{\pm0.6}$	$23_{\pm0.8}$	$0.84_{\pm0.009}$	$0.18_{\pm0.02}$
DeepSeek-7B	$6_{\pm0.6}$	$39_{\pm1.0}$	$12_{\pm0.6}$	$23_{\pm0.8}$	$0.83_{\pm0.010}$	$0.17_{\pm0.02}$
GenEDA-1B	$8_{\pm0.7}$	$34_{\pm1.2}$	$13_{\pm0.8}$	$20_{\pm0.9}$	$0.90_{\pm0.011}$	$0.45_{\pm0.03}$
GenEDA-7B	$11_{\pm0.8}$	$41_{\pm1.3}$	$14_{\pm0.8}$	$\mathbf{23}_{\pm0.9}$	$0.92_{\pm0.011}$	$0.47_{\pm0.03}$
NetlistFormer-1B	$\mathbf{12}_{\pm0.8}$	$\mathbf{44}_{\pm1.4}$	$\mathbf{16}_{\pm0.9}$	$20_{\pm1.0}$	$\mathbf{0.93}_{\pm0.010}$	$\mathbf{0.49}_{\pm0.03}$
NetlistFormer-7B	$\mathbf{14}_{\pm0.9}$	$\mathbf{46}_{\pm1.4}$	$\mathbf{16}_{\pm0.9}$	$\mathbf{24}_{\pm1.1}$	$\mathbf{0.93}_{\pm0.009}$	$\mathbf{0.50}_{\pm0.02}$

converted into graph representations as described in Section 3. We randomly select 400 subcircuits for evaluation that are excluded from the training data. Because no public benchmark currently provides paired netlist–specification annotations for these tasks, we construct new text labels to support both training and evaluation. [1]. Since formal RTL specifications are seldom released alongside public hardware designs, following [13, 26] we instead generate the ground-truth references by prompting GPT-4o with each design's RTL code. This RTL code is used solely for ground-truth construction and is never provided to the models under evaluation. Therefore, NetlistFormer and all baselines generate their predictions from the gate-level netlist alone.

Baselines. We evaluate NetlistFormer against a diverse set of LLM baselines to ensure a thorough comparison. The general-purpose baselines range from large commercial systems such as GPT-4o and DeepSeek-V3 to lightweight open-source alternatives, namely DeepSeek-Coder at 1B and 7B scales. We further include GenEDA at 1B and 7B scales, which adopts an MLP-based connector to bridge the netlist encoder and the LLM decoder. For each baseline, the model receives the textual representation of the netlist together with a question prompt as input, and is required to produce the corresponding specification text.

Implementation Details. For the encoder in NetlistFormer, we use NetTAG [12], a state-of-the-art netlist encoder, as the backbone. For the decoder, we fine-tune the DeepSeek-Coder [17] 1B and 7B models. The training pipeline incorporates both DeepSpeed ZeRO and LoRA [18] to improve efficiency and scalability. We initialize the self-attention layers with pretrained BERT [9] weights, while the cross-attention layers are randomly initialized. The entire transformer network, including the queries, contains a total of 188 million parameters. In our experiments, we use 32 queries, each with a dimension of 768. All experiments are conducted on 4 NVIDIA RTX A6000 (48 GB) GPUs, and all reported results are averaged over three independent runs with different random seeds.

[1]We provide the dataset containing the generated ground truth text labels at https://anonymous.4open.science/r/Generated_Label-4C0C

5.2 Specification Reasoning

Evaluation Protocol. For evaluation, we directly compare the generated natural language specifications with the ground-truth texts, using both natural language similarity metrics and LLM-assisted functional assessment metrics. In particular, we report standard n-gram-based similarity scores, namely BLEU and ROUGE-1/2/L, which quantify the degree of overlap between the generated and ground-truth texts based on n-gram matching. To further assess semantic consistency, we adopt an embedding-based evaluation strategy using the text-embedding-ada-002 model. This approach encodes both generated and ground-truth specifications into vector representations and then computes cosine similarity to measure their semantic alignment. In addition, we use GPT-o1 to directly evaluate the specifications, assigning a similarity score between 0 and 1 based on how well the generated specification reflects the intended functionality of the design.

Experiment Results. Table 1 reports the specification reasoning performance on Task 1 and Task 2. Both NetlistFormer-1B and NetlistFormer-7B, which are built on DeepSeek-Coder at the 1B and 7B scales, achieve substantially better scores than almost all general-purpose LLMs across all metrics. In particular, on Task 1, NetlistFormer shows substantially larger improvements over the baseline LLMs across multiple metrics. This suggests that, because Task 1 requires global information to describe circuit functionality, allowing the queries to selectively attend to the netlist for text generation enables NetlistFormer to capture the overall design intent

Model Output:

(1) **Interface**: The module is designed to perform a division operation, likely used in a processor or similar system. The inputs include a clock signal, a reset signal, and a few control and status signals related to the division process. The output signal, div_resHi, is likely used for a subsequent operation.
(2) **Purpose**: The module appears to be part of a larger system that performs division operations, likely within a processor or similar architecture.
(3) **Functionality**: Module processes a division operation by performing a series of logical operations and updates the output register 'div_resHi' based on the input signals. Key states or operations include: - The module checks for a reset conditioned updates the 'div_resHi' based on the 'div/mulReg' value.
(4) **Constraints**: The module uses a clock signal for synchronous operations, and a reset signal to initialize the 'div_resHi' value.

Ground Truth:

(1) **Interface**: Inputs: `clk`, `rst`, `div_state`, `div_count`, `div_isHi`, `ex_ctrl_div`, `ex_reg_valid` - Outputs: `div_resHi`
(2) **Purpose**: This Verilog module appears to be part a division operation unit, managing the readiness and validity of division requests and maintaining the result state `div_resHi`.
(3) **Functionality**: - The module facilitates the control logic for division operations by determining request readiness and processing division result states. - Key operations include checking the division state and count, evaluating validity and readiness condition and updating the division result state.
(4) **Constraints**: - There is a clock signal (`clk`) for sequential logic. - While a `rst` signal is mentioned as input, it doesn't appear to be used in the provided code for resetting the module's state.

Model Output:

(1) **Interface**: Inputs: clock, reset, IR, datai. Output: wr.
(2) **Purpose**: This module manages the write signal according to internal logic. Operation is synchronous with the positive edge of the clock.
(3) **Functionality**: The module processes a combination of input signals to determine the next value of the output register, wr. Key states or operations include: Conditional assignment of the output register based on the input data and the reset signal. The output register is updated on the positive edge of the clock signal.
(4) **Constraints**: The module is reset to a known value on the positive edge of the clock signal. The reset signal is used to initialize the output register to a known value.

Ground Truth:

(1) **Interface**: - Inputs: `clk`, `rst`, `state`- Output: `wr`.
(2) **Purpose**: - The module is designed to control a write signal (`wr`) based on the conditions derived from internal signal computations. It operates synchronously on the positive edge of a clock signal (`clk`).
(3) **Functionality**:- The module evaluates a combination of conditions from its internal signals and updates the output register `wr`. Key operations/conditions include checking whether a portion of an 8-bit signal (`_122_`) equals `7`, manipulating negated states, and gated signal flows.
(4) **Constraints**:- The module uses a clock input (`clk`) to synchronize operations and is presumably initialized or reset with a reset input (`rst`), or specified in the given code. However, the reset functionality itself is not explicitly implemented

(a) Task1 - Functional Description

Model Output:

(1) **Combinational Logic Computations**: The module contains a significant amount of combinatorial logic for determining the state and handling exceptions. This includes decoding instructions, evaluating branching conditions, or CSR command processing with checks against.
(2) **Sequential Register Update Function**:- An always block is present that updates 'wb_reg_xcpt' based upon computation from other signals. Uses clock information (`clk`) along side various intermediary signal computations which influence this register update process.
(3) **State Machine Or Pipeline Flow:** No explicit FSMs are defined; however, there appears to be implicit flow through input instruction combinations such as checking validity states before executing operations., Exception Handling involves complex decision making around specific exception types using control inputs related specifically towards breakpoints, interrupts & trap causes via BPU interfaces etc.).

Ground Truth:

(1) **Combinational Logic Computations**: - The module heavily utilizes combinational logic for determining the state of exceptions, valid instructions, and control signals for status registers. This logic includes calculating the result of bitwise operations involving control registers and input signals, deciding on branching decisions.
(2) **Sequential Register Update Function**: - Sequential logic is used to update state on the `clk` signal, most notably updating the `wb_reg_xcpt` register based on exception conditions expressed in combinational logic.
(3) **State Machine or Pipeline Flow:** - No explicit FSM is defined; however, the module is part of a pipeline, reacting dynamically to various inputs signifying different stages of the execution cycle. The pipeline handles data serialization, identification & handling of various exceptions, executing control status register operations, and employing breakpoints while maintaining flow continuity with proper handling of exceptions.

Model Output:

(1) **Combinational Logic Computations:** Hazard detection is performed using address matching (`HazardSimplePlugin_addr0Match` and `HazardSimplePlugin_addr1Match`). Control signals like whether the shifter should be active, based on conditions of other plugins such as shift control signal. The logic involves multiple conditional checks to determine if operations are valid or require stalling.
(2) **Sequential Register Update Function:** An always block synchronously updates 'execute_LightShifterPlugin_isActive' with a new value determined by combinatorial computations. Uses clock input ('clk') for this sequenced update process.
(3) **State Machine or Pipeline Flows:** If any explicit state machines aren't evident from code snippet provided here then there isn't an extensive set up involving states directly in module implementation.

Ground Truth:

(1) **Combinational Logic Computations**: Hazard detection through address matching (`HazardSimplePlugin_addr0Match`, `HazardSimplePlugin_addr1Match`) and use signals. Condition logic to determine light shifter activity and hazard status. Logic to decide if the light shifter operation has been completed.
(2) **Sequential Register Update Function:** The `execute_LightShifterPlugin_isActive` register is updated on the rising edge of `clk`, based on the combinational logic that determines if the shifter operation should remain active (`_0057_`).
(3) **State Machine or Pipeline Flow:** This module appears to be part of a pipeline stage handling execution, focusing on shifting operations and managing data hazards. No explicit state machine is present, but it incorporates hazard management logic and the light shifter control which may affect the state of computation in the pipeline.

(b) Task2 - Implementation Detail

Figure 4: Comparison between text generated from NetlistFormer and ground truth text.

more effectively than the baseline LLMs. Moreover, scaling from NetlistFormer-1B to NetlistFormer-7B leads to consistent improvements across all metrics, which suggests that stronger open-source backbone models offer a promising direction for further performance enhancement.

To complement these quantitative results with a qualitative perspective, we provide visualizations in Figure 4 to compare the model-generated text with the corresponding ground-truth text. We examine how closely the model predictions align with the ground-truth text in (a) Task 1 — Functional Description and (b) Task 2 — Implementation Detail. The generated texts successfully capture the key specification details of the module and show strong agreement with the ground-truth text. For example, in the left case of Figure 4(a), the model correctly identifies that the module is responsible for performing a division operation (shown in orange), clearly describes the main input interfaces, including the clock and reset signals (shown in pink), and accurately characterizes the role of the output signal div_resHi (shown in blue and green), all of which align with the ground-truth description. However, one observed tendency is that the model sometimes abstracts certain detailed input signals into more general expressions, such as "several control and status signals," rather than explicitly enumerating them. Nevertheless, considering the four dimensions of interface, purpose, functionality, and constraints as a whole, the generated specification demonstrates a high degree of semantic consistency with the ground-truth specification.

5.3 Keywords Evaluation

Evaluation Protocol. While the n-gram, embedding-based, and GPT-assisted scores in Section 5.2 capture overall textual and semantic similarity, they are not specifically designed to assess the technical content of circuit specifications. In practice, hardware descriptions can contain template phrases, such as *Interface* and *Purpose*, as well as common English function words, which can inflate surface-level overlap regardless of whether the model correctly identifies the relevant signals, modules, or behaviors. To complement the standard metrics with a domain-aware perspective, we introduce three additional metrics, *Keyword F1*, *Bigram F1*, and *Trigram F1*, that explicitly measure how faithfully the generated text reproduces the technical vocabulary and phrasing of the ground-truth.

To quantify these metrics, we first construct a domain dictionary from the ground-truth corpus. We aggregate specifications across all datasets, and tokenize each text after lowercasing and removing English stop words, and template words. From the resulting token sequences, we extract unigrams, consecutive bigrams, and consecutive trigrams, and retain only those whose corpus frequency exceeds a threshold of 100. This frequency filter retains only the

Table 2: Keywords Evaluation on Task 1 (Functional Description) and Task 2 (Implementation Detail).

Model	Task 1: Functional Description			Task 2: Implementation Detail		
	Keyword F1	Bigram F1	Trigram F1	Keyword F1	Bigram F1	Trigram F1
GPT4o	$8.73_{\pm 0.9}$	$3.32_{\pm 0.5}$	$4.03_{\pm 0.6}$	$11.78_{\pm 1.1}$	$6.26_{\pm 0.7}$	$6.39_{\pm 0.7}$
DeepSeek-V3	$11.31_{\pm 1.1}$	$6.64_{\pm 0.8}$	$7.49_{\pm 0.9}$	$14.77_{\pm 1.3}$	$7.33_{\pm 0.9}$	$7.13_{\pm 0.8}$
DeepSeek-1B	$15.29_{\pm 1.4}$	$10.14_{\pm 1.1}$	$2.04_{\pm 1.5}$	$15.96_{\pm 1.4}$	$7.33_{\pm 0.9}$	$18.92_{\pm 2.0}$
DeepSeek-7B	$17.25_{\pm 1.6}$	$5.52_{\pm 0.7}$	$4.95_{\pm 0.7}$	$16.00_{\pm 1.5}$	$9.25_{\pm 1.1}$	$7.86_{\pm 0.9}$
GenEDA-1B	$9.56_{\pm 1.5}$	$3.64_{\pm 0.8}$	$12.42_{\pm 1.6}$	$30.74_{\pm 2.3}$	$14.20_{\pm 1.7}$	$14.85_{\pm 1.7}$
GenEDA-7B	$19.55_{\pm 2.0}$	$9.62_{\pm 1.3}$	$\mathbf{14.16}_{\pm 1.7}$	$33.18_{\pm 2.5}$	$16.92_{\pm 1.9}$	$17.20_{\pm 1.9}$
NetlistFormer-1B	$\mathbf{36.57}_{\pm 2.7}$	$\mathbf{15.59}_{\pm 1.8}$	$12.42_{\pm 1.6}$	$\mathbf{36.86}_{\pm 2.7}$	$\mathbf{18.45}_{\pm 2.0}$	$\mathbf{18.05}_{\pm 2.0}$
NetlistFormer-7B	$\mathbf{38.39}_{\pm 2.8}$	$\mathbf{16.42}_{\pm 1.9}$	$\mathbf{12.74}_{\pm 1.6}$	$\mathbf{37.62}_{\pm 2.8}$	$\mathbf{18.83}_{\pm 2.7}$	$\mathbf{22.30}_{\pm 2.3}$

Table 3: Ablation Studies of each component in NetlistFormer (Specification Reasoning on Task 1).

Model	BLEU	ROUGE-1	ROUGE-2	ROUGE-L	Emb. Sim.	GPT Score
w/o $\mathcal{L}_{\text{NTC}}$	$20_{\pm 1.7}$	$52_{\pm 2.6}$	$21_{\pm 1.8}$	$30_{\pm 2.1}$	$0.83_{\pm 0.020}$	$0.40_{\pm 0.05}$
w/o $\mathcal{L}_{\text{NTM}}$	$23_{\pm 1.9}$	$53_{\pm 2.7}$	$25_{\pm 2.0}$	$34_{\pm 2.3}$	$0.90_{\pm 0.022}$	$0.48_{\pm 0.05}$
w/o $\mathcal{L}_{\text{NTG}}$	$18_{\pm 1.6}$	$49_{\pm 2.5}$	$18_{\pm 1.6}$	$28_{\pm 2.0}$	$0.89_{\pm 0.022}$	$0.36_{\pm 0.04}$
w/o Q	$15_{\pm 1.4}$	$45_{\pm 2.3}$	$14_{\pm 1.4}$	$25_{\pm 1.9}$	$0.79_{\pm 0.025}$	$0.30_{\pm 0.04}$
w/o $\mathcal{L}_{\text{RG}}$	$10_{\pm 1.0}$	$38_{\pm 2.0}$	$9_{\pm 1.0}$	$20_{\pm 1.6}$	$0.85_{\pm 0.020}$	$0.28_{\pm 0.03}$
NetlistFormer	$\mathbf{24}_{\pm 2.0}$	$\mathbf{57}_{\pm 2.9}$	$\mathbf{26}_{\pm 2.1}$	$\mathbf{35}_{\pm 2.4}$	$\mathbf{0.93}_{\pm 0.020}$	$\mathbf{0.50}_{\pm 0.04}$

key recurring domain-specific vocabulary that characterizes circuit specification language. The resulting dictionary forms a fixed reference set of circuit-relevant unigrams, bigrams, and trigrams.

Given the dictionary, each metric is computed as a multiset precision/recall/F1 over the filtered token sequences. Let C_p and C_r be the token Counters of the prediction and ground-truth. The overlap is computed as $|C_p \cap C_r| = \sum_t \min(C_p(t), C_r(t))$, giving $P = |C_p \cap C_r|/|C_p|$, $R = |C_p \cap C_r|/|C_r|$, and $F_1 = 2PR/(P + R)$. *Keyword F1* measures whether the model uses the correct circuit-domain single words, while *Bigram F1* and *Trigram F1* extend this evaluation to short word combinations (e.g., *state machine reset signal*) and commonly used expressions (e.g., *finite state machine active high reset*), which capture longer phrase patterns that often appear in hardware specifications.

Experiment Results. Table 2 reports the keyword-level evaluation on Task 1 and Task 2. Across both tasks, NetlistFormer achieves the highest scores on nearly all F1 metrics, demonstrating that our models reproduce the domain-specific vocabulary of circuit specifications far more faithfully than the baselines. This indicates that the gains observed in Section 5.2 are not driven by template phrases or generic words, but reflect a genuine improvement in the model's ability to identify circuit-relevant signals, modules, and behaviors. Finally, scaling from NetlistFormer-1B to NetlistFormer-7B yields consistent gains across nearly all metrics, further showing that NetlistFormer benefits steadily from stronger backbones.

5.4 Ablation Study on Components of NetlistFormer

Table 3 reports the ablation study results analyzing the effect of each major component of NetlistFormer on functional description performance. The *All* setting, which uses all components together, achieves the best performance across all evaluation metrics, indicating that the components contribute to the model in a complementary manner.

The largest degradation is observed in the *w/o* $\mathcal{L}_{\text{RG}}$ setting. This shows that alignment learning in stage 1 alone is insufficient for task-oriented text generation, and that stage 2 instruction tuning plays an important role in specification generation. The next largest drop occurs in the *w/o* Q setting, demonstrating that learnable queries are important for selectively extracting circuit information required for text generation and delivering it to the LLM. Among the three stage 1 losses, the performance impact is largest for *w/o* $\mathcal{L}_{\text{NTG}}$, followed by *w/o* $\mathcal{L}_{\text{NTC}}$ and *w/o* $\mathcal{L}_{\text{NTM}}$. This suggests that $\mathcal{L}_{\text{NTG}}$ contributes most directly to learning how to map circuit structures into natural language, whereas $\mathcal{L}_{\text{NTM}}$ has the smallest effect because its learning signal partially overlaps with those of the contrastive losses. Notably, removing any single Stage 1 objective leads to substantial degradation across all metrics, e.g., w/o $\mathcal{L}_{\text{NTG}}$ drops BLEU by 25% and GPT Score by 28%. This degradation is more pronounced than what is typically observed for individual objective ablations in vision-language Q-Former pre-training, suggesting that the netlist–text abstraction gap demands a stronger combination of global, token-level, and generative supervision than the image–caption setting. In other words, multi-objective alignment is not a stylistic choice inherited from BLIP-2 but a domain-level necessity for circuit-to-text generation.

6 Conclusion

In this paper, we propose NetlistFormer, a cross-modal circuit transformer that bridges netlist structure and natural language through a query-based two-stage framework. In contrast to prior MLP-based connectors that rely solely on autoregressive supervision, NetlistFormer adopts a multi-objective cross-modal alignment strategy that jointly optimizes contrastive, matching, and netlist-grounded generation objectives. Through this strategy, the learnable queries learn to extract text-guided structural information relevant to description generation, and are projected into an LLM representation space for circuit-to-text fine-tuning. Experiments on both functional description and implementation detail tasks show that NetlistFormer consistently outperforms general-purpose LLMs and MLP-based connector baselines across standard similarity metrics as well as domain-aware keyword-level metrics. To the best of our knowledge, NetlistFormer is the first circuit generative transformer trained with multi-objective cross-modal alignment. We believe this fine-grained circuit–text alignment provides a promising foundation for future circuit foundation models that unify structural reasoning and natural-language generation.

References

[1] Christoph Albrecht. 2005. IWLS 2005 benchmarks. In *International Workshop for Logic Synthesis*.

[2] Lilas Alrahis, Abhrajit Sengupta, Johann Knechtel, Satwik Patnaik, Hani Saleh, Baker Mohammad, Mahmoud Al-Qutayri, and Ozgur Sinanoglu. 2021. Gnn-re: graph neural networks for reverse engineering of gate-level netlists. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41, 8, 2435–2448.

[3] Alon Amid et al. 2020. Chipyard: integrated design, simulation, and implementation framework for custom socs. *IEEE Micro*, 40, 4, 10–21.

[4] Kaiyan Chang, Ying Wang, Haimeng Ren, Mengdi Wang, Shengwen Liang, Yinhe Han, Huawei Li, and Xiaowei Li. 2023. Chipgpt: how far are we from natural language hardware design. *arXiv preprint arXiv:2305.14019*.

[5] Lei Chen et al. 2024. Large circuit models: opportunities and challenges. *Science China Information Sciences*, 67, 10, 200402.

[6] Subhajit Dutta Chowdhury, Kaixin Yang, and Pierluigi Nuzzo. 2021. Reignn: state register identification using graph neural networks for circuit reverse engineering. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 1–9.

[7] Fulvio Corno, Matteo Sonza Reorda, and Giovanni Squillero. 2002. RT-level ITC'99 benchmarks and first ATPG results. *IEEE Design & Test of Computers*, 17, 3, 44–53.

[8] Chenhui Deng, Zichao Yue, Cunxi Yu, Gokce Sarar, Ryan Carey, Rajeev Jain, and Zhiru Zhang. 2024. Less is more: hop-wise graph attention for scalable and generalizable learning on circuits. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 1–6.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

[10] Wenji Fang, Wang Jing, Yao Lu, Shang Liu, and Zhiyao Xie. 2025. GenEDA: towards generative netlist functional reasoning via cross-modal circuit encoder-decoder alignment. In *Proceedings of the 44th IEEE/ACM International Conference On Computer-Aided Design*.

[11] Wenji Fang, Mengming Li, Min Li, Zhiyuan Yan, Shang Liu, Hongce Zhang, and Zhiyao Xie. 2024. AssertLLM: generating and evaluating hardware verification assertions from design specifications via multi-LLMs. *arXiv preprint arXiv:2402.00386*.

[12] Wenji Fang, Wenkai Li, Shang Liu, Yao Lu, Hongce Zhang, and Zhiyao Xie. 2025. NetTAG: a multimodal RTL-and-layout-aligned netlist foundation model via text-attributed graph. In *Proceedings of the 62nd ACM/IEEE Design Automation Conference*.

[13] Wenji Fang, Shang Liu, Jing Wang, and Zhiyao Xie. 2025. Circuitfusion: multimodal circuit representation learning for agile chip design. *arXiv preprint arXiv:2505.02168*.

[14] Wenji Fang, Shang Liu, Hongce Zhang, and Zhiyao Xie. 2025. A self-supervised, pre-trained, and cross-stage-aligned circuit encoder provides a foundation for various design tasks. In *Proceedings of the 30th Asia and South Pacific Design Automation Conference*.

[15] Wenji Fang, Jing Wang, Yao Lu, Shang Liu, Yuchao Wu, Yuzhe Ma, and Zhiyao Xie. 2025. A survey of circuit foundation model: foundation ai models for VLSI circuit design and EDA. *arXiv preprint arXiv:2504.03711*.

[16] Adria Gascón, Pramod Subramanyan, Bruno Dutertre, Ashish Tiwari, Dejan Jovanović, and Sharad Malik. 2014. Template-based circuit understanding. In *2014 Formal Methods in Computer-Aided Design (FMCAD)*. IEEE, 83–90.

[17] Daya Guo et al. 2024. DeepSeek-Coder: when the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

[18] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: low-rank adaptation of large language models. In *The 10th International Conference on Learning Representations*.

[19] Rahul Kande, Hammond Pearce, Benjamin Tan, Brendan Dolan-Gavitt, Shailja Thakur, Ramesh Karri, and Jeyavijayan Rajendran. 2024. (security) assertions by large language models. *IEEE Transactions on Information Forensics and Security*, 19, 4374–4389.

[20] Sadaf Khan, Zhengyuan Shi, Ziyang Zheng, Min Li, and Qiang Xu. 2025. DeepSeq2: enhanced sequential circuit learning with disentangled representations. In *Proceedings of the 30th Asia and South Pacific Design Automation Conference*, 498–504.

[21] Andreas Kuehlmann, Viresh Paruthi, Florian Krohm, and Malay K Ganai. 2002. Robust boolean reasoning for equivalence checking and functional property verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 21, 12, 1377–1394.

[22] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. 2023. BLIP-2: bootstrapping language-image pre-training with frozen image encoders and large language models. In *International Conference on Machine Learning*.

[23] Min Li, Sadaf Khan, Zhengyuan Shi, Naixing Wang, Huang Yu, and Qiang Xu. 2022. DeepGate: learning neural representations of logic gates. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*.

[24] Wenchao Li, Zach Wasson, and Sanjit A Seshia. 2012. Reverse engineering circuits using behavioral pattern mining. In *2012 IEEE international symposium on hardware-oriented security and trust*. IEEE, 83–88.

[25] Mingjie Liu et al. 2023. Chipnemo: domain-adapted llms for chip design. *arXiv preprint arXiv:2311.00176*.

[26] Yi Liu, Changran Xu, Yunhao Zhou, Zeju Li, and Qiang Xu. 2025. DeepRTL: bridging verilog understanding and generation with a unified representation model. *arXiv preprint arXiv:2502.15832*.

[27] Charles Papon and HDL Spinal. 2024. VexRiscv, a FPGA friendly 32 bit RISC-V CPU implementation.

[28] Zehua Pei, Hui-Ling Zhen, Mingxuan Yuan, Yu Huang, and Bei Yu. 2024. Betterv: controlled verilog generation with discriminative guidance. *arXiv preprint arXiv:2402.03375*.

[29] Zhengyuan Shi, Ziyang Zheng, Sadaf Khan, Jianyuan Zhong, Min Li, and Qiang Xu. 2024. DeepGate3: towards scalable circuit representation learning. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*.

[30] Zhengyuan Shi et al. 2023. DeepGate2: functionality-aware circuit representation learning. In *Proceedings of the 42nd IEEE/ACM International Conference on Computer-Aided Design*.

[31] Pramod Subramanyan, Nestan Tsiskaridze, Kanika Pasricha, Dillon Reisman, Adriana Susnea, and Sharad Malik. 2013. Reverse engineering digital circuits using functional analysis. In *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 1277–1280.

[32] Chuyue Sun, Christopher Hahn, and Caroline Trippel. 2023. Towards improving verification productivity with circuit-aware translation of natural language to systemverilog assertions. In *First International Workshop on Deep Learning-aided Verification*.

[33] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30*.

[34] Ziyi Wang, Chen Bai, Zhuolun He, Guangliang Zhang, Qiang Xu, Tsung-Yi Ho, Bei Yu, and Yu Huang. 2022. Functionality matters in netlist representation learning. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 61–66.

[35] Clifford Wolf, Johann Glaser, and Johannes Kepler. 2013. Yosys-a free verilog synthesis suite. In *Proceedings of the 21st Austrian Workshop on Microelectronics*.

[36] Haoyuan Wu, Haisheng Zheng, Yuan Pu, and Bei Yu. 2025. Circuit representation learning with masked gate modeling and verilog-aig alignment. *arXiv preprint arXiv:2502.12732*.

[37] Nan Wu, Yingjie Li, Cong Hao, Steve Dai, Cunxi Yu, and Yuan Xie. 2023. Gamora: graph learning based symbolic reasoning for large-scale boolean networks. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 1–6.

[38] Ceyu Xu, Pragya Sharma, Tianshu Wang, and Lisa Wu Wills. 2023. Fast, robust and transferable prediction for hardware logic synthesis. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, 167–179.

[39] Xufeng Yao, Haoyang Li, Tsz Ho Chan, Wenyi Xiao, Mingxuan Yuan, Yu Huang, Lei Chen, and Bei Yu. 2025. Hdldebugger: streamlining hdl debugging with large language models. *ACM Transactions on Design Automation of Electronic Systems*, 30, 6, 1–26.

[40] Xufeng Yao, Yiwen Wang, Xing Li, Yingzhao Lian, Ran Chen, Lei Chen, Mingxuan Yuan, Hong Xu, and Bei Yu. 2024. RTLRewriter: methodologies for large models aided rtl code optimization. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, 1–7.

[41] Ziyang Zheng, Shan Huang, Jianyuan Zhong, Zhengyuan Shi, Guohao Dai, Ningyi Xu, and Qiang Xu. 2025. DeepGate4: efficient and effective representation learning for circuit design at scale. *arXiv preprint arXiv:2502.01681*.