GPTAC: Domain-Specific Generative Pre-Trained Model for Approximate Circuit Design Exploration

Sipei Yi, *Student Member, IEEE*, Weichuan Zuo, Hongyi Wu, Ruicheng Dai, *Member, IEEE*, Weikang Qian, *Senior Member, IEEE*, and Jienan Chen, *Senior Member, IEEE*

***Abstract*—Automatically designing fast and low-cost digital circuits is challenging because of the discrete nature of circuits and the enormous design space, particularly in the exploration of approximate circuits. However, recent advances in generative artificial intelligence (GAI) have shed light to address these challenges. In this work, we present GPTAC, a domain-specific generative pre-trained (GPT) model customized for designing approximate circuits. By specifying the desired circuit accuracy or area, GPTAC can automatically generate an approximate circuit using its generative capabilities. We represent circuits using domain-specific language tokens, refined through a hardware description language keyword filter applied to gate-level code. This representation enables GPTAC to effectively learn approximate circuits from existing datasets by leveraging the GPT language model, as the training data can be directly derived from gate-level code. Additionally, by focusing on a domain-specific language, only a limited set of keywords is maintained, facilitating faster model convergence. To improve the success rate of the generated circuits, we introduce a circuit check rule that masks the GPTAC inference results when necessary. The experiment indicated that GPTAC is capable of producing approximate multipliers in under 15 seconds while utilizing merely 4GB of GPU memory, achieving a 10-40% reduction in area relative to the accuracy multiplier depending on various accuracy needs.**

***Index Terms*—Generative artificial intelligence (GAI), domain-specific generative pre-trained (GPT) model, approximate circuit design.**

I. Introduction

AUTOMATICALLY designing digital circuits is a critical and highly beneficial task in modern electronic design, offering significant advantages in terms of design efficiency, performance optimization, and reduced human error. As the demand for smaller, faster, and more energy-efficient circuits grows, automated tools that can rapidly generate complex designs have become increasingly essential. However, automatic circuit design is a challenging task, particularly when it comes to approximate circuits. These circuits are designed by relaxing the exact functional requirements, aiming to reduce hardware cost such as area and power consumption. Compared to accurate circuit design, the introduction of accuracy as an additional design consideration significantly expands the design space, hence increasing the complexity of the design process. Moreover, an effective approximate circuit must achieve a balanced trade-off among accuracy, area, and power consumption, making the design process even more challenging. Thus, the development of effective and efficient methods for approximate circuit design remains a key research area in electronic design automation (EDA).

Some existing approaches to approximate circuit design rely primarily on the fixed structures of the target designs to simplify the exploration of trade-offs among accuracy, area, and performance. Notable examples of these methods include truncation and rounding. While they are effective in reducing hardware cost, they typically lack flexibility, as they can only handle circuits with the specific fixed structures. Several studies [1], [2], [3], [4] ave exploited these approaches in different application domains, but they often fail to provide an effective solution for various circuit designs. Consequently, there is growing interest in exploring more flexible design methods, which can automate the design of approximate circuits more effectively and efficiently. One such method is the recently proposed approximate logic synthesis method based on Versatile Efficiency-Accuracy Configurable Batch Error Estimation (VECBEE) [5], which is applicable to any statistical error measurement and any graph-based circuit representation. While it is flexible and can accelerate the error estimation process, its reliance on greedy search and extensive Monte Carlo simulations limits its overall efficiency and solution quality, leaving room for further improvement.

Meanwhile, the recent development of generative artificial intelligence (GAI) has shed light on circuit design and optimization. With the development of the GAI technology, the use of machine learning methods can substantially improve circuit design, particularly in circuit optimization [6], [7]. Various learning models are used as surrogate models to evaluate designs during the optimization process [8], [9], [10]. For example, in [8], an active learning-based framework was proposed to optimize prefix adders. It uses a regression model

Received 20 December 2024; revised 10 March 2025; accepted 30 April 2025. Date of publication 9 May 2025; date of current version 16 June 2025. This work was supported by the National Natural Science Foundation of China under Grant 62331009. This article was recommended by Guest Editor L. Zhang. *(Corresponding author: Jienan Chen.)*

Sipei Yi, Weichuan Zuo, and Jienan Chen are with the National Key Laboratory of Wireless Communications, University of Electronic Science and Technology of China, Chengdu, Sichuan 611731, China (e-mail: Jesson.Chen@outlook.com).

Hongyi Wu is with the School of Automation and Information Engineering, Sichuan University of Science and Engineering, Zigong, Sichuan 643002, China.

Ruicheng Dai is with the University of Michigan-Shanghai Jiao Tong University Joint Institute, Shanghai 200240, China.

Weikang Qian is with the University of Michigan-Shanghai Jiao Tong University Joint Institute, Shanghai 200240, China, and also with the MoE Key Laboratory of Artificial Intelligence, Shanghai Jiao Tong University, Shanghai 200240, China.

Digital Object Identifier 10.1109/JETCAS.2025.3568606

based on the Gaussian process to predict delay and area using features extracted from the prefix tree structure. Building upon this, Geng et al. proposed a sequential optimization framework that enables automatic feature learning for prefix adder structures [9]. This framework uses a graph neural network (GNN) as a surrogate model, facilitating more efficient and effective exploration of adder structures.

However, machine learning-aided circuit design faces several challenges, with the primary hurdle being the effective representation of circuits. Current approaches often rely on graph-based representations, where circuit elements (such as gates and wires) are modeled as nodes and edges in a graph [11], [12], [13], [14], [15], [16]. This allows for efficient encoding of circuit structures and the interconnections, making it suitable for machine learning algorithms. For example, IronMan, an end-to-end graph-based circuit design framework, aims to enable flexible and automated design space exploration (DSE), providing optimized solutions under user-specified constraints [12], [13]. DeepGate2 [14] and DeepGate3 [15] employ an efficient one-round GNN for circuit design. Furthermore, Liu et al. proposed a GNN-based method for Boolean representation to optimize circuits [15], and Shi et al. combine a GNN with a deep Q-learning network (DQN) for improved circuit optimization [16]. However, graph-based circuit learning methods struggle with structural adaptability. When the circuit structure changes, the graph-based representation becomes difficult to automatically reconstruct due to the variation in circuit structure. This makes structure generation a particularly challenging task for graph-based methods.

Another approach for representing circuits is to directly use large language models (LLMs) [7], [17], [18], [19]. For example, ChatEDA is an autonomous agent for EDA powered by an LLM [17]. In [7], a generative pre-trained (GPT) approach is introduced, which generates prompts for the LLM to produce initial Verilog codes. In [18], a framework is proposed to democratize the design of AI accelerators leveraging natural language. However, the challenge with LLMs is that they are general models, making it difficult to generate domain-specific circuits. For example, while an LLM can generate a general multiplier, it struggles to produce a multiplier that meets specific optimization goals.

In this work, we propose GPTAC, a domain-specific generative pre-trained model for approximate circuit design. To address the circuit representation challenge, we introduce a domain-specific language token to represent circuits. GPTAC can efficiently generate approximate circuits with specific area and accuracy requirements from a large dataset of learned circuit designs. Our contributions are summarized as follows:

- **Domain-Specific Generative Pre-Trained Model for Approximate Circuit Design.** GPTAC utilizes the GPT-Transformer learning model to learn the features of approximate circuits. Circuits are represented using domain-specific language tokens, which provide flexibility in circuit representation and transformation. By optimizing the model structure and training methods, GPTAC can automatically generate circuits with the desired area and accuracy. This model enables systematic exploration of the optimization space for 8-bit signed approximate multipliers.
- **Token-Refined Domain-Specific Circuit Representation.** To facilitate circuit representation, we refine domain-specific language tokens by applying a hardware language keyword filter to gate-level code. This representation enables GPTAC to effectively learn approximate circuits from existing datasets, leveraging the LLM-GPT model, as the training data can be directly derived from gate-level code. Focusing on a domain-specific language ensures that only a limited set of keywords is maintained, enabling faster model convergence.
- **Optimized Model Design and Fine-Tuning Methods.** We employ a decoder-only GPT transformer to learn approximate circuits. The accuracy and area of the circuit are rapidly evaluated by a simulation tool, and the results are integrated with the refined circuit tokens as inputs to GPTAC. A fine-tuning method is applied to enhance the model's ability to handle specific tasks.
- **Design Rule Check (DRC) Masking Regime.** To improve and guarantee circuit correctness, we propose a design rule check (DRC) mask regime during the generation of the circuit tokens. The DRC mask checks the rationality of gate tokens, wire connectivity, and grammar, masking invalid designs.

The experiment demonstrated that GPTAC can generate approximate multipliers in less than 15 seconds while only using 4GB of GPU memory, resulting in a 10-40% decrease in area compared to an accuracy multiplier, depending on different accuracy requirements.

The remainder of the paper is structured as follows. Section II provides the background. Section III outlines the GPTAC framework. Section IV introduces the circuit representation and the dataset generation process. Section V delves into the network architecture of GPTAC and the model training. Section VI details the approximate circuit generation procedure. Section VII shows the experimental results. Finally, Section VIII concludes the paper.

II. Background

A. Transformer-Based Neural Network

In the realm of LLM, three architectures predominate: encoder-only architectures, decoder-only architectures, and encoder-decoder architectures. BERT exemplifies the encoder-only architecture, while Google T5 [20] serves as an example of the encoder-decoder architecture. The GPT series, however, consistently adopts the decoder-only architectures.

An encoder-only language model differs from decoder-only and encoder-decoder types, which are autoregressive. Encoder-only models focus on comprehending the input to produce task-specific outputs. Under the same depth, the encoder-decoder architecture has twice the number of network layers compared to the other two architectures. Considering both computational efficiency and general applicability, decoder-only architecture emerges as a preferable option. The study in [21] compares the training effectiveness of prevalent models. It finds that a causal decoder-only architecture with the full language modeling objective exhibits superior zero-shot generalization capability, provided that the large model is pre-trained in an unsupervised way. This observation can be partially explained by theoretical reasoning.

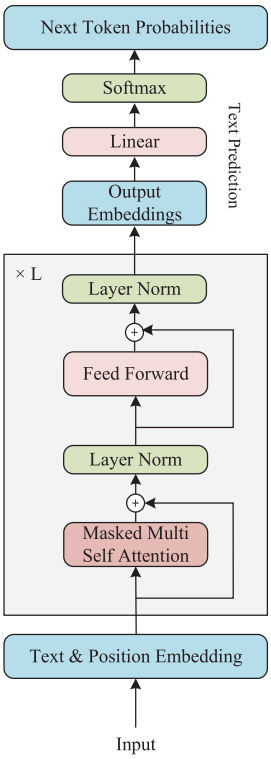

Fig. 1. Transformer decoder architecture of GPT.

As shown in Fig. 1, in text prediction, the aim is to determine the probability distribution of the given text sequence. For the input sequence of tokens $u_1, u_2, \ldots, u_n$, the resulting output from the language model is:

$$p(u_1, u_2, \ldots, u_n) = p(u_1) \prod_{i=2}^{n} p(u_i | u_1, u_2, \ldots, u_{i-1}). \quad (1)$$

Consider the single-head model as an illustrative example. Given an input sequence with n tokens, each of dimension d, denoted as $X \in R^{d \times n}$, the self-attention is calculated as follows:

$$\text{Attention}(X) = W_v X \, \text{Softmax}\left[\frac{(W_k X)^T (W_q X)}{\sqrt{d_k}}\right], \quad (2)$$

where $W_q \in \mathbb{R}^{d_q \times d}$, $W_k \in \mathbb{R}^{d_k \times d}$, and $W_v \in \mathbb{R}^{d_v \times d}$. Since the number of heads is 1, we have $d_q = d_k = d_v = d$.

The Softmax component in Eq. (2) corresponds to the attention matrix P, given by:

$$P = \text{Softmax}\left[\frac{(W_k X)^T (W_q X)}{\sqrt{d_k}}\right]. \quad (3)$$

$P \in \mathbb{R}^{n \times n}$ captures the relationships between words. In practice, the relationships between words in text are complex, so there must exist appropriate matrices W_q and W_k to convert X into P as shown in Eq. (3). Failure to achieve this reduces the expressiveness of the model, making training less effective. In [22], it shows that a necessary condition for the existence of such a solution is $d \geq n$. Regarding the encoder, its two-way attention mechanism can lead to a low-rank issue. In contrast, the decoder, which employs an unidirectional attention, forms a triangular array that is more likely to be of full rank, potentially enhancing its expressiveness.

B. Approximate Circuits

Approximate circuit design can be divided into three main categories based on the design layers: algorithm, architecture, and circuit [23], [24]. At the algorithm layer, the research in [2] refines the multiplication tasks into shifts and additions, aiming to lower the energy used in the computations, which is ideal for neural network applications where a large error is tolerable. In terms of floating-point computations, the work [25] proposes a linear fitting strategy combined with error management to reduce energy consumption, while the work [26] introduces the RMAC technique that enables flexible precision tuning by adding mantissas.

At the architectural level, the work [27] focuses on reducing hardware complexity and energy consumption through the static segment method applied to the input stage. The work [28] introduces the under-designed multiplier (UDM) strategy tailored for non-error-tolerant applications by integrating partial product correction. Furthermore, the work [29] improves 4-2 compressor designs by incorporating error recovery modules, which considerably decreases hardware cost and power consumption.

At the circuit level, Boolean rewriting techniques have been applied to approximate multipliers to further reduce power consumption [28], [29]. Additionally, [30] adopts an evolutionary algorithm to develop EvoApprox8b, a library of approximate circuits with different trade-offs among accuracy, area, and delay.

As artificial intelligence becomes more prevalent, machine learning-based approaches for automated design have attracted increasing interest. Cartesian genetic programming is applied in [31] to enhance circuit design and augment the EvoApprox8b library, although its ability to handle complex circuits is still limited. Machine learning techniques are used in [32] for faster exploration of the design space, whereas the work [33] introduces a hybrid integer genetic algorithm for optimizing partial product compression, aiming to reduce hardware cost.

Another approach for automated approximate circuit design is through approximate logic synthesis (ALS), in which many methods are greedy. For example, ALSRAC [34] generates high-quality approximate circuits through the use of approximate care sets and the resubstitution techniques, achieving significant reduction in hardware cost. However, the error estimation approach in ALS faces challenges in simultaneously achieving high accuracy and efficiency, particularly for large designs. To address this, VECBEE [5] introduces a method to accelerate the error estimation process while maintaining the accuracy. It not only improves the circuit quality but also significantly accelerates the process, demonstrating superior results on multiple benchmark circuits.

III. Overview of GPTAC

The workflow of GPTAC is divided into three main phases: 1) circuit representation and dataset generation, 2) model training, and 3) approximate circuit generation, as shown in Fig. 2. The dataset generation phase involves the creation and augmentation of domain-specific data, tailored to encapsulate the details and variability of various circuit designs. This phase ensures that the data not only reflect current standards, but also

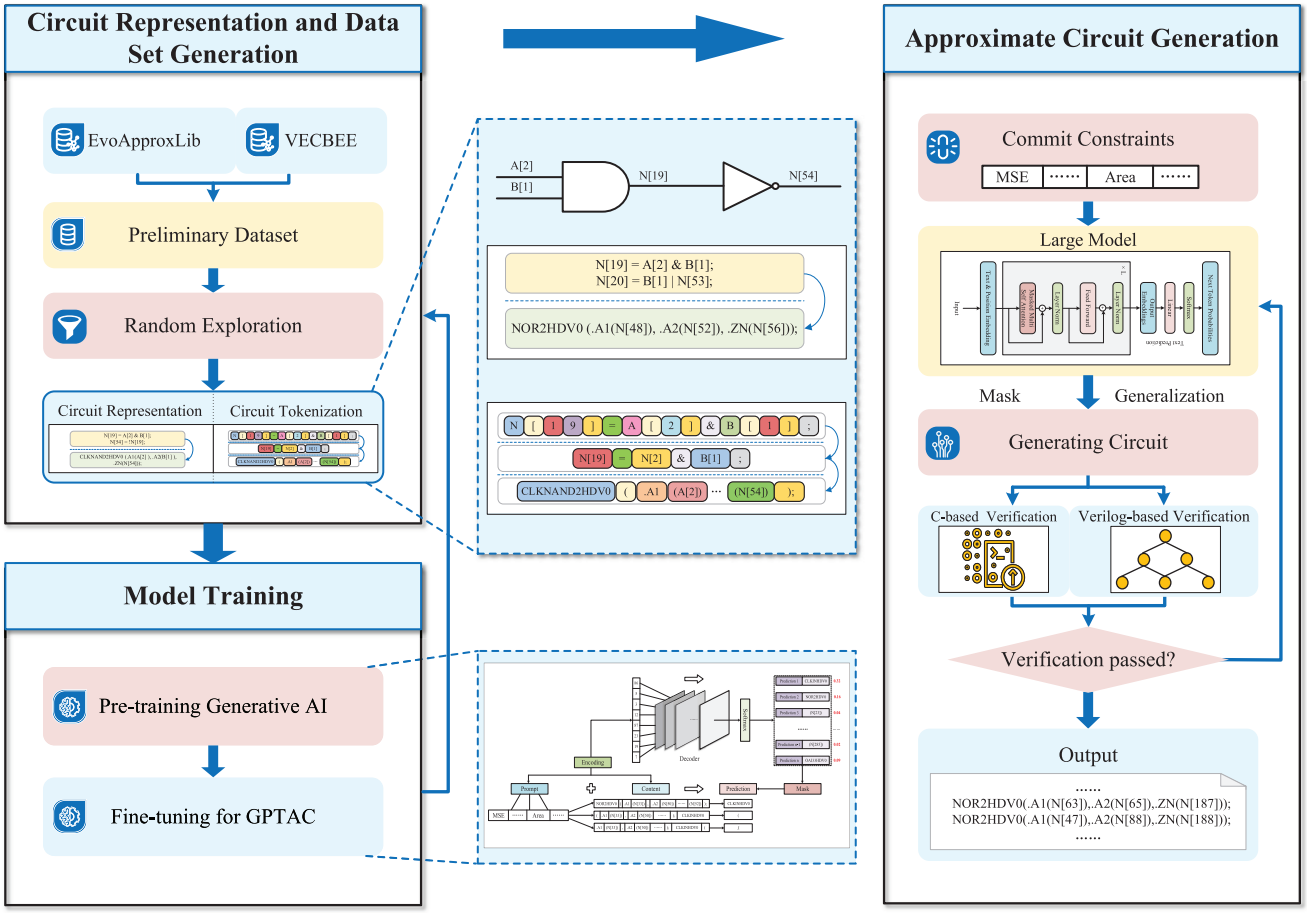

Fig. 2. The training and inference workflow of the GPTAC domain-specific model: the workflow sequentially progresses from left to right through training, output generation, and validation/feedback iteration.

incorporate a diverse range of modifications and enhancements to support robust model learning.

After the dataset is prepared, the model training phase begins. Here, the model undergoes an initial phase of self-supervised learning, where it learns to recognize and predict patterns and structures inherent to circuit design without explicit external labels. This is followed by a fine-tuning process, where the model's parameters are adjusted and optimized based on a smaller, more specific set of training data. This phase is crucial to refining the ability of the model to generate precise and functional circuits.

After rigorous training and validation of the accuracy and reliability of the model, it progresses to the approximate circuit generation phase. In this final stage, GPTAC employs a sophisticated interface that integrates with the output circuit token. This mechanism enables the model to generate the desired approximate circuits based on the specifications and constraints from the user. The integration of the output token with the model's inference engine enables the generation of circuits that not only meet the desired functional requirements but also optimize for parameters such as area, delay, and power. This phase is the culmination of the model's capabilities, which demonstrates its practical utility to produce viable, efficient, and innovative circuit solutions.

For error characterization, we adopt Mean Squared Error (MSE) in Eq. (4) as the primary metric, leveraging second-order moment analysis to effectively capture significant statistical deviations—a methodology aligned with state-of-the-art approximate circuit frameworks like VECBEE [5]. The sample generation phase employs embedded C scripts for automated circuit code expansion and simulation, integrating syntax verification and structural integrity checks while constraining MSE estimates within 2% tolerance. Behavioral-level simulations subsequently validate precise error metrics. Performance evaluation implements a two-stage methodology: Preliminary area estimation via integer-normalized cell summation based on SMIC 55nm process design kit (PDK) enables rapid model training, while Synopsys Design Compiler-based implementation provides final metrics (physical area, power consumption, critical path delay) for hardware-accurate benchmarking.

$$\text{MSE} = \frac{1}{N}\sum_{i=1}^{N}(\text{approximate}_i - \text{accurate}_i)^2. \tag{4}$$

The system's operational objective is to explore approximate circuit designs that simultaneously satisfy the MSE

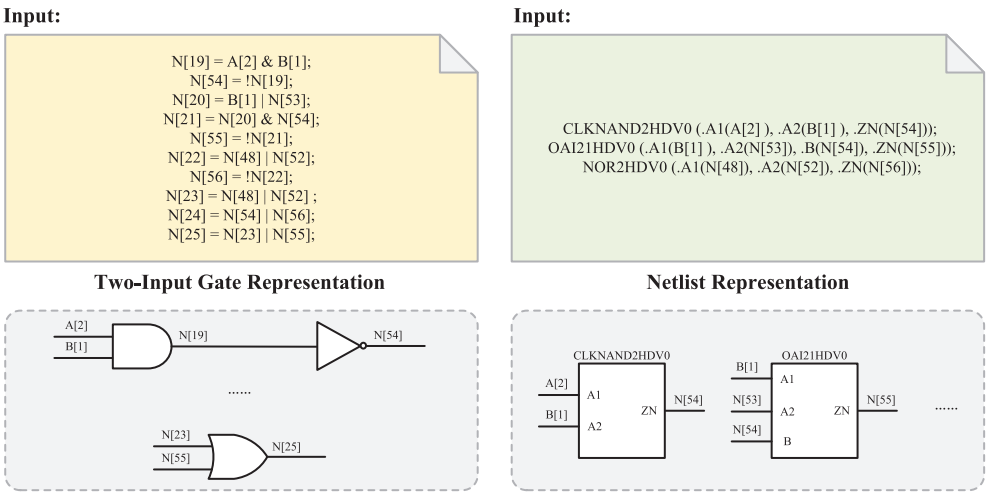

Fig. 3. The gate and netlist representation (Left: 2-input logic gate representation; Right: Standard cell implementation).

constraint with minimal deviation and achieve the lowest estimated area under the given MSE bound.

IV. Circuit Representation and Dataset Generation

Initially, we gather data on approximate multipliers from EvoApproxLib [30], a publicly available library for approximate computing, to form the dataset. However, the amount of available data in the library is limited. To extend the dataset, we employ the VECBEE [5] synthesis method for additional data augmentation, creating our preliminary dataset. To further improve the data quality, we implement a random exploration, which modifies the base dataset through random deletion and modification of gates, resulting in an expanded dataset.

We then proceed to standardize the circuits necessary for constructing the dataset intended to train the GPT model. For circuit representation, the Verilog language is utilized first. Then, we evaluate the model's training performance by comparing two data formats: logic expressions using two-input gates and those based on standard cells after converting the HDL into a gate-level netlist format, as shown in Fig. 3.

The first option, with full expansion, entails a greater amount of text but captures simpler graph connections akin to a binary tree. In contrast, the second option reduces text volume by utilizing multiple-input gates, but leading to more complex graph connections.

Furthermore, it is important to include the MSE and circuit area as prefix information in the dataset. MSE is determined by comparing the approximate and exact outputs of the circuit. Area can be efficiently estimated by summing the areas of all gates in the circuit.

For the tokenization strategy, we employ two techniques illustrated in Fig. 4: single-character slicing and circuit-node slicing. The single-character slicing method divides the expression into its individual characters, which minimizes the dictionary size but loses the graph structure information of the circuit. In contrast, the circuit-node slicing approach considers signals, ports, module names, logical operators, and formatting symbols as nodes within the dictionary. Although it increases the total number of entries in the dictionary, it retains the graph structure information of the circuit.

The impact of various circuit representations and tokenization techniques on the performance of model training is evaluated, with detailed results presented in the experimental section. Subsequently, we proceed to tokenize the gate-level netlist. The netlist includes a considerable amount of redundant linguistic information, making the direct application of the GPT LLM ineffective and hindering the convergence. Therefore, we refine it by excluding non-essential keywords that do not impact the circuit, followed by tokenizing the pertinent keywords. We then designate module and wire keywords as individual tokens, which helps shorten the prediction length and enhance the standard semantics. As illustrated in Fig. 2, each keyword in the netlist is tokenized, and only the terms relevant to the circuit are retained. This process produces a domain-specific dataset enabling efficient model training.

Note that we will modify the data selection in accordance with the network training performance, balancing the trade-off between specialization and generalization. Thus, the dataset evolves through an iterative process.

V. GPTAC Network Architecture and Model Training

To exploit GAI in designing approximate circuits through textual circuit descriptions, two main strategies are suggested for task formulation. The first strategy treats it as an outcome-oriented task following particular problem specification, similar to machine translation, aligning with the Transformer model that includes both encoder and decoder components. The second strategy approaches it as a text prediction task based on initial conditions, consistent with the decoder-only Transformer architecture used extensively in LLMs like GPT.

While both architectures are generally effective, they have limitations in exploring approximate circuit design. The encoder-decoder setup excels in tasks where input

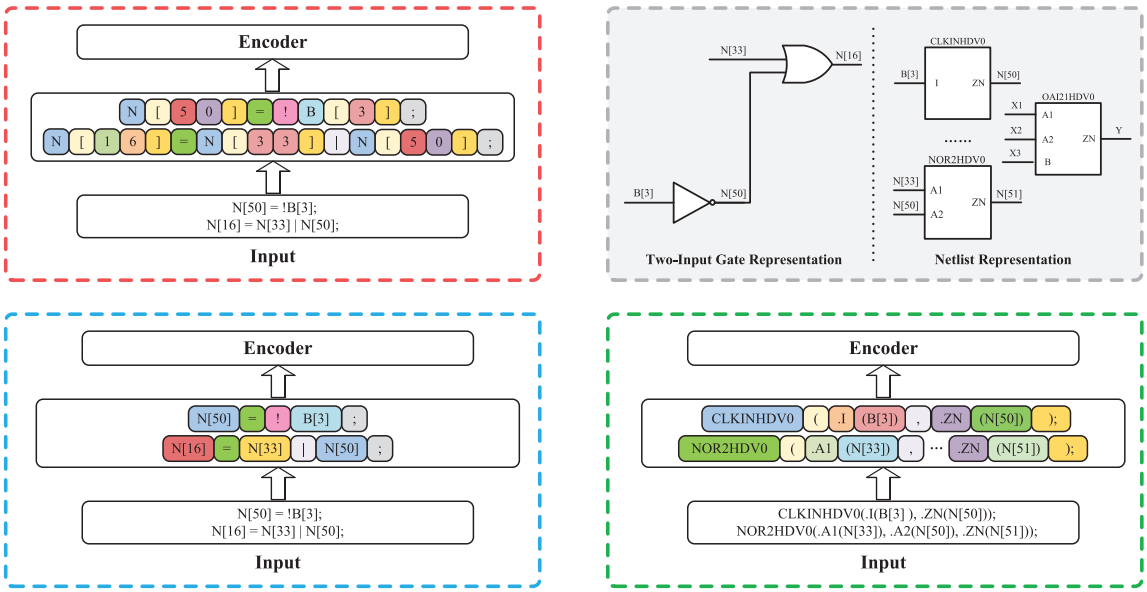

Fig. 4. The tokenization process (Top-left: Character-level vectorization; Bottom-left/right: Node-level vectorization for logic-gates/standard-cell-netlist).

and output requirements are relatively balanced, such as in machine translation. However, in our scenario, the data includes brief constraints along with complex circuit encodings, resulting in multiple potential solutions for a single constraint, thereby complicating direct one-to-one mappings. The encoder-decoder can comprehend broad text relationships via global self-attention, performing optimally when input and output complexities are similar. However, with simple inputs and complex outputs, this architecture may use resources inefficiently and introduce redundant parameters, especially in the encoder component.

The decoder-only model, commonly employed in text prediction frameworks such as GPT, encounters a different issue. In our context, constraints are located at the start of the circuit text and impact the output only when they remain within the prediction window. As the window progresses past the location of the constraints, the linkage to them diminishes, and the relationship between the constraints and circuit text weakens. Moreover, the autoregressive nature of the decoder-only model, where each step is contingent on the prior output, struggles to preserve robust context. This poses significant challenges in lengthy texts with complex constraints, as subsequent tokens exhibit reduced responsiveness to initial inputs.

In summary, it is essential to create a neural network architecture capable of delivering context-aware inference for extensive texts while effectively transforming input instructions to meet the requirements of approximate circuit exploration tasks. For this purpose, we propose an instruction-enhanced decoder-only model, which involves a slight modification but results in significant performance improvement.

Specifically, throughout both the training and inference stages, we incorporate the constraint instructions into the moving prediction window, ensuring that each window integrates these constraints while concealing the constraint text in the final output to preserve syntactic accuracy. As shown in Fig. 5, before the encoding phase, the prediction window advances with the constraint instruction remaining at the starting point. This ensures its persistent role as explicit guidance in subsequent inferences. This technique capitalizes on the advantages of a decoder-only framework in text prediction tasks, while preserving a robust connection between the generated text and the constraints. We evaluated the performance of three distinct structural models via pre-training, with related data presented in the experimental section.

VI. Approximate Circuit Generation

Once the GPTAC model has undergone fine-tuning, we can provide it with a prompt specifying the desired area or accuracy, allowing it to automatically generate the circuit tokens. During the token generation process, we utilize a probability distribution-based expansion sampling method to enhance the diversity of the circuits produced.

Although large models demonstrate impressive generalization abilities, even the top-performing model in our study—which has been trained on gate-level netlists and tokenized by circuit nodes. GPTAC employs an instruction-enhanced decoder-only configuration, which can occasionally generate circuits containing syntax errors such as broken lines or duplicate assignments. To mitigate this, we implement a specialized mask mechanism for the output phase, acting as a filter designed to rectify circuit errors using a design rule checking tool. This approach aims to impose constraints on the composition of circuit expressions in the output vector, thereby preventing incorrect assignment of output signals to input ports, avoiding repeated assignments of the same signal to output ports, and ensuring the complete presence of signals at input ports. Further details can be found in Algorithm 1. Specifically, we first initialize the output signal set by incorporating the input ports. Subsequently, during the model-based circuit generation process, we examine the properties of the

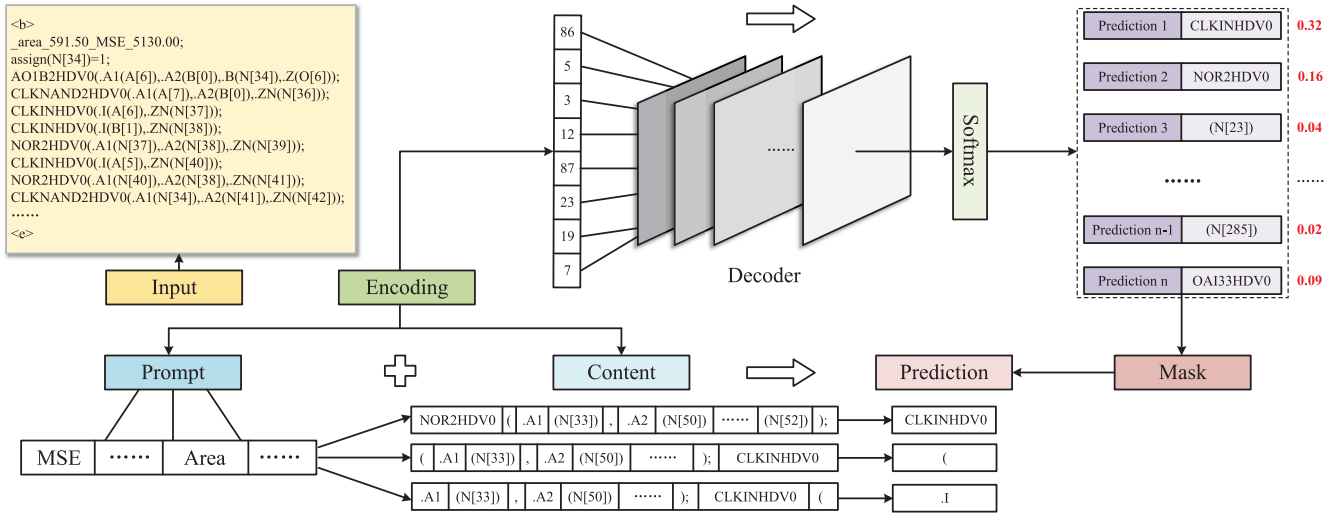

Fig. 5. Neural network architecture and data flow specifically designed for approximate circuit exploration task.

Algorithm 1 The Mask Mechanism.

Require: P_o: probability distribution of the prediction; M: maximum number of words;
Ensure: R: The correct circuit code;
Initialize output signal set $\mathcal{S} \leftarrow \{\text{Port}_{in}\}$;
for $i$in range$(0, M)$ **do**
  while $x \leftarrow$ Sampling by P_o **do**
    if $x$is an input signal **then**
      if $x$not in $\mathcal{S}$ **then**
        Set the probability of x to zero;
      else
        Push x to R;
        break;
      end if
    else if $x$is an output signal **then**
      Push $x$to $\mathcal{S}$and R;
    else if $x$does not conform to syntax **then**
      Set the probability of x to zero;
    end if
  end while
end for

Algorithm 2 Generalization Enhancement Algorithm.

Require: P_o: probability distribution of the prediction; $\mathcal{A}$: approximate substitution set; λ: randomness factor
Ensure: Enhanced output variety
while $R$is not over **do**
  $x \leftarrow$ Sampling from P_o
  $l \leftarrow$ Sample uniformly from $[0, 1]$
  if $l < \lambda$ **then**
    if $l < \frac{\lambda}{2}$ **then**
      Replace previous cell using $\mathcal{A}$
    else
      Set probability of x to zero
      continue▷Resampling
    end if
  end if
  if $x$does not cause an error **then**
    Push x to R
  end if
end while

circuit. Specifically, we modify the output probability distribution and perform resampling whenever open circuits or language errors occur, until a correct circuit is obtained.

Furthermore, to improve the generalizability of the output results, we introduce a generalization exploration method applied during the output sampling phase. One simple approach is to sequentially replace gates in the pre-output results with the second most likely vector from the probability distribution. This subtle modification triggers a chain reaction in subsequent text predictions, producing a variety of diverse approximate circuits. The details of the implementation can be found in Algorithm 2. Specifically, we construct an approximation set $\mathcal{A}$ by representing some cells as their approximate counterparts. Subsequently, we achieve a balance in randomness and select the approximation method through uniform sampling. Specifically, if the uniform sampling value is less than λ, we replace the last cell with its substitute; otherwise, we set the prediction's probability to zero and continue sampling. Through the self-correction mechanism, the model is able to generate new viable circuits.

VII. Experimental Results

A. Experimental Setup

At present, our data is based on the 8-bit signed multiplier. All the system demonstrates deployability on cost-efficient hardware configurations, with detailed specifications of computational resources, software toolchain versions, and employed PDK systematically documented in Table I. This implementation substantiates the framework's adaptability for resource-constrained environments while maintaining design

TABLE I
CONFIGURATION OF THE EXPERIMENTAL PLCATFORM

Experimental Parameters	
CPU	Intel i9-14900KF
GPU	NVIDIA RTX4090D 24GB
RAM	DDR5-5200 128GB
Operating System	Ubuntu 24.04.2 LTS
Synopsys Design Compiler	S-2021.06-SP5
Process Design Kit	SMIC 55nm PDK
Simulation Clock Cycle Constraints	$20ns$

integrity. During the pre-training stage, a dataset comprising 43,000 unfiltered circuit samples generated by VECBEE [5] was utilized. This was augmented with 3,000 additional circuit samples produced through random and greedy search techniques, derived from the EvoApprox8b [30] dataset. For the training and evaluation processes, the dataset was randomly partitioned into a 9:1 ratio. In the fine-tuning phase, 700 circuit samples were carefully selected from the total of 46,000, ensuring high precision and optimal performance to enhance the model's accuracy concerning performance constraints in output circuits. The training loss $L(\theta)$ and accuracy are determined according to Eq. (5) and Eq. (6).

$$L(\theta) = -\frac{1}{N}\sum_{i=1}^{N} logP(y_i|x_i,\theta), \tag{5}$$

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions}, \tag{6}$$

where θ denotes the model parameters, N indicates the number of samples, and $P(y_i|x_i,\theta)$ represents the predicted probability for the i-th instance by the model.

To achieve the optimal training results, it is essential to adjust these parameters dynamically. A larger model does not always yield better performance due to two primary reasons: limited computational resources, and the risk of overfitting when the input size is small and the task is relatively simple, which results in reduced generalization performance. This issue is more noticeable in deeper neural networks. Thus, balancing the depth of the network is crucial, along with increasing the number of attention heads and their dimensions to enhance multi-layered attention. This strategy aids in effectively capturing shared features in high-quality circuit data for improved outputs. Through extensive testing, we identified the optimal parameters for pre-training and fine-tuning. Table IV lists the detailed information on model parameters, training settings, durations, evaluation loss, and accuracy for pre-training and fine-tuning phases.

B. Experimental Analysis and Comparison

We investigate four alternative approaches, which result from combining two circuit representation formats (two-input logic gates and standard cells) with two tokenization strategies (single-character and circuit-node based tokenization). Each approach was incorporated into our neural network framework to evaluate the training performance, as detailed in Table II.

TABLE II
TRAINING PERFORMANCE WITH DIFFERENT DATA FORMATS AND TOKENIZATION METHODS

Performance		Tokens per sample	Syntax error rate	Circuit error rate
Tow-input logic gate	Single character	26531	62.10%	93.36%
	Circuit node	5192	23.16%	86.72%
Standard cells	Single character	16578	89.32%	31.65%
	Circuit node	3215	31.65%	46.93%

TABLE III
PERFORMANCE COMPARISON OF DIFFERENT NEURAL NETWORK ARCHITECTURES

Performance	Model architecture		
	Encoder + Decoder	Decoder-only	GPTAC
Model parameters	201.88M		
Evaluation accuracy (24h pre-training)	67.92%	97.67%	99.68%
Error output (syntax & circuit)	86.31%	46.19%	23.61%
Output variation per input token	13.85%	9.63%	21.37%

The evaluation metrics include the syntax error rate of the resulting circuits calculated by Eq. (7) and the circuit error rate, specifically addressing broken lines and multiple assignments, calculated by Eq. (8).

$$Syntax\ error\ rate = \frac{1}{N}\sum_{i=1}^{N} P(Syntax\ Error|x_i,\theta), \tag{7}$$

$$Circuit\ error\ rate = \frac{1}{N}\sum_{i=1}^{N} P(Circuit\ Error|x_i,\theta), \tag{8}$$

where θ represents the model parameters, N denotes the number of samples, and x_i is the limiting words.

Based on the experimental results shown in Table II, we chose the expression based on the standard cells combined with node-based tokenization as the optimal training resource for the domain-specific large model.

We implemented training for large models across three different neural network architectures, with an equal number of parameters for each. To evaluate their performance, we sampled 1,000 outputs and summarized the results in Table III. Our assessment concentrated on three key metrics: accuracy as defined in Eq. (6), syntactic correctness, and the impact of approximate constraints on output variability. This impact is measured by the average percentage of token deviations in the output concerning its total length, stemming from alterations to a single input token.

We compared the approximate multipliers from the EvoApprox8b library, derived from the VECBEE synthesis tool, and generated by GPTAC through batch generalization. For each MSE constraint, we selected representative circuit samples for

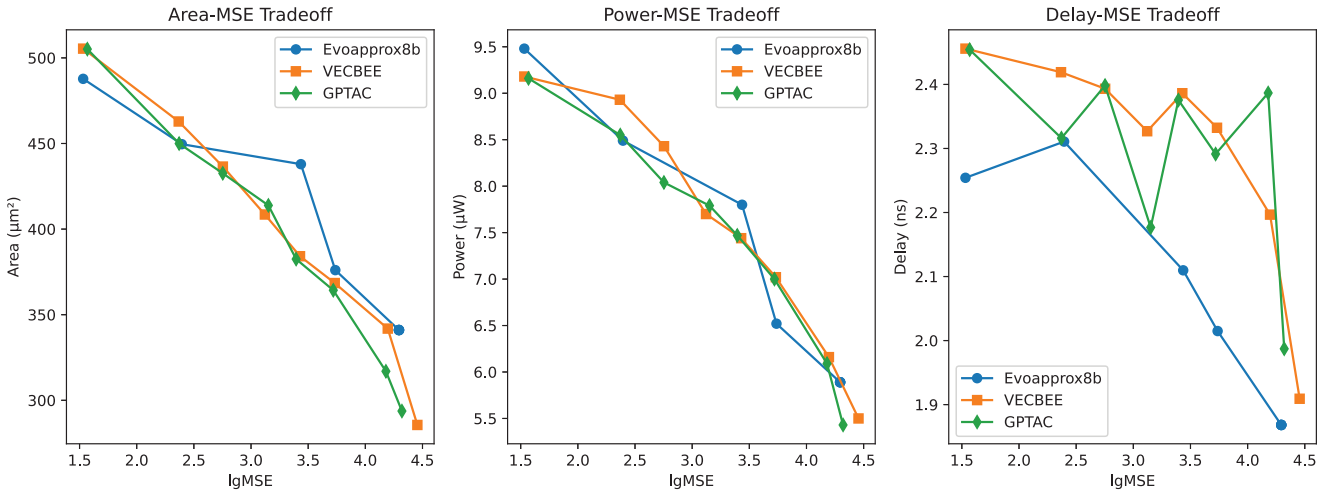

Fig. 6. Performance-error trade-off plot of sampled designs from Evoapprox8b, VECBEE, and GPTAC.

TABLE IV
DETAILED PARAMETERS FOR GPTAC MODEL PRE-TRAINING AND FINE-TUNING

Detailed Model Training Parameters (on RTX4090D*1)		
stages of training	**Pre-training**	**Fine-turning**
n_layer	16	
n_head	32	
n_embd	1024	
parameters	201.88M	
block_size	1024	
batch_size	32	31
Dataset size	46000	700
eval_iters	1600	
weight_decay	1e-1	
beta1	0.9	
beta2	0.95	
grad_clip	1.0	
warmup_iters	9600	0
Accuracy of constraint instruction	57%	100%
dropout	0.2	0.3
Gradient Accumulation	1	
max learning rate	5e-4	1e-4
minimum learning rate	5e-5	1e-5
VRAM usage druing training	24127MB	23201MB
Time to stable loss	23h	17h
Stable evaluation accuracy	99.68%	99.85%
VRAM usage during inference	3980MB	
Inference speed	15s per sample (4096 tokens)	

accuracy evaluation and performance comparison, as outlined in Table V. All tabular data presented in this work have been published in our GitHub repository: https://github.com/YiRen19/GPTAC-finished.git, where we have additionally open-sourced the model implementation codes. Additionally, performance-error trade-off line plots for the tabular samples are visualized in Fig. 6 to enable intuitive comparative analysis. The results show that when the MSE constraint is more than 500, designs produced by both VECBEE and GPTAC considerably outperform those from EvoApprox8b. The experiment demonstrated that GPTAC is reducing area by 10-40% compared to the accuracy multiplier, contingent on diverse accuracy requirements.

For MSE constraints under 5000, GPTAC achieves performance comparable to VECBEE, with a difference within 1%. However, for MSE constraints over 5000, GPTAC increasingly outperforms VECBEE. With an MSE constraint of 30,000, circuits generated by GPTAC achieve a 26.7% improvement in MSE over VECBEE under nearly area, along with a 41% reduction in area compared to the exact 8-bit signed multiplier.

With an MSE approaching 15,000, GPTAC-generated samples exhibit a 7.3% area advantage over VECBEE. As demonstrated by the partial circuit analysis in Fig. 7, during high-bit computations (e.g., the most significant bit O[15]), GPTAC achieves computational results nearly identical to VECBEE samples while employing fewer logic units. This optimization reduces the total cell count, thereby decreasing both circuit area and power consumption.

Trained on extensive datasets of approximate circuits, GPTAC efficiently identifies hidden characteristics of top-quality approximate circuits. By improving the model's generalization and output potential, GPTAC finds approximate circuits that are closer to the global optimal solution. For the 8-bit signed multiplier, when the MSE constraint surpasses 5000, GPTAC effectively explores optimization spaces that the greedy strategy neglects.

Furthermore, when the MSE constraints are set below 5000, GPTAC reliably produces several circuits that nearly match VECBEE's performance. Among these, a few circuits demonstrate an area advantage ranging from 0.3% to 1% under the same MSE conditions. Although their performance is similar, these circuits display significant structural variability, showcasing GPTAC's capability to

TABLE V

COMPARISON OF APPROXIMATE MULTIPLIERS FROM EVOAPPROX8B LIBRARY, PRODUCED BY VECBEE TOOL, AND PRODUCED BY GPTAC

Output Results Comparison	Constraint Value	Performance of Approximate Multiplier Simulated using Synopsys Design Compiler with the SMIC 55nm PDK.											
		Evoapprox8b				VECBEE				GPTAC			
Constraints: MSE	N	Area(μm^2)	Power(μW)	Delay(ns)	MSE	Area(μm^2)	Power(μW)	Delay(ns)	MSE	Area(μm^2)	Power(μW)	Delay(ns)	MSE
	30	487.76	9.48	2.25419	34	505.40	9.18	2.45591	34	505.12	9.16	2.45432	37
	300	449.6	8.49	2.31074	248	462.84	8.93	2.41909	233	449.96	8.55	2.31613	235
	500	—	—	—	—	436.52	8.43	2.39327	567	432.60	8.04	2.39831	566
	1400	—	—	—	—	408.52	7.70	2.32694	1322	413.84	7.79	2.17669	1419
	3000	437.92	7.80	2.10971	2731	384.16	7.44	2.38628	2682	382.48	7.47	2.37563	2481
	5000	376.04	6.52	2.01487	5462	368.48	7.02	2.33234	5398	364.28	7.00	2.29133	5246
	15000	—	—	—	—	**341.88**	**6.16**	**2.19667**	**15709**	**316.96**	**6.09**	**2.38666**	**15131**
	20000-30000	341.04	5.89	1.86821	19690	**285.60**	**5.50**	**1.90915**	**28517**	**293.72**	**5.43**	**1.98717**	**20899**
	The area/power/delay of the exact 8-bit signed array multiplier is 496.44μm^2/9.10μW/2.04371ns with MSE = 0.												

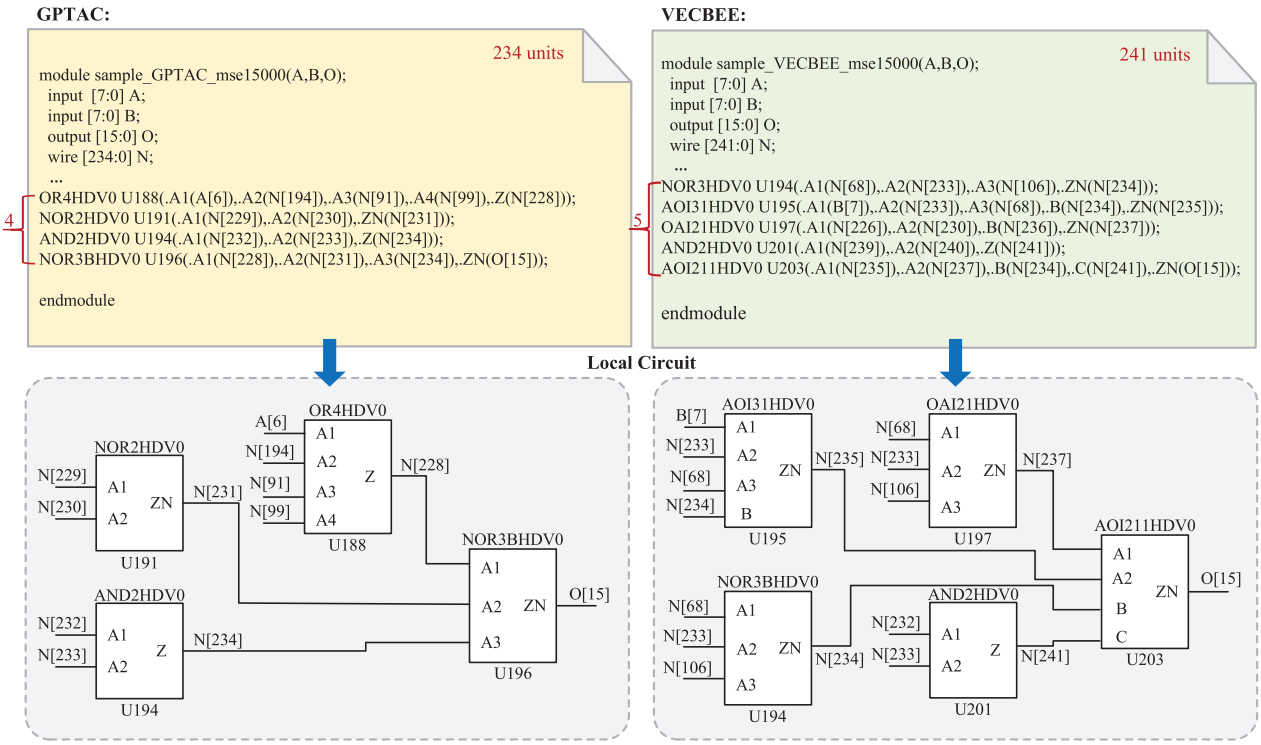

Fig. 7. Comparative analysis of localized structural discrepancies between GPTAC- and VECBEE-generated specimens under an MSE constraint of 15,000.

explore a wider range of optimization options in approximate circuit design compared to VECBEE.

Although GPTAC incurs 7.5% higher latency than Evoapprox8b under MSE constraints, it achieves superior area efficiency (9.9% improvement) and lower power consumption with MSE constraints over 3000. This latency-area trade-off aligns with low-power and low-area design priorities. Crucially, GPTAC supports continuous gradient-based MSE-constrained optimization, unlike the discontinuous MSE gradient characteristics of Evoapprox8b.

Notably, the GPTAC-generated sample with MSE in 5246 exhibits an area occupation of 364.28μm^2, demonstrating comparable post-simulation area characteristics to the open-source 8-bit approximate multiplier from the DeBAM [3] framework. A comprehensive comparison of these two multipliers across multiple metrics is presented in Table VII. The analysis reveals that GPTAC achieves superior power efficiency and significant MSE advantages at the cost of partial timing performance degradation. However, DeBAM demonstrates distinct superiority in MAE metrics. This mutual exclusivity in optimizing different objectives highlights the critical challenge of designing models capable of holistically balancing diverse error and performance metrics. Developing such multi-objective optimization frameworks to attain balanced Pareto-optimal solutions constitutes a primary objective of our future research agenda.

Moreover, we assessed the practical performance of VECBEE and GPTAC when implemented on desktop computing systems in Table I. GPTAC delivered notably superior performance relative to VECBEE running in a single-thread mode. Additionally, for 8-bit signed multipliers, under low MSE constraints, the search space for greedy optimization narrows, resulting in a reduced number of unique and valid samples. In contrast, with high MSE constraints, such as those

TABLE VI
COMPARISON OF RUNNING PERFORMANCE BETWEEN VECBEE SYNTHESIS TOOL AND GPTAC DOMAIN-SPECIFIC LARGE MODEL

Performance Analysis of 8-bit signed Approximate Circuits		VECBEE [5]		GPTAC
	(N, M)	Single-threaded	32-threaded	
Usable data generated per hour with $N < MSE < M$	(0,100)	15	23	31
	(100,1000)	43	96	56
	(1000,10000)	51	132	62
	(10000,20000)	25	48	51
	(20000,30000)	17	26	37
Computational Resource Overhead		RAM 17.3GB	RAM 62.5GB	RAM 4.6GB + VRAM 3.97GB

TABLE VII
DEBAM VS. GPTAC UNDER AREA PROXIMITY TO 365 μm^2

Performance	DeBAM	GPTAC
Area(μm^2)	368.76	364.28
Power(μW)	7.25	7
Delay(ns)	1.71154	2.29133
MSE	747139	5246
MAE	333	554

exceeding 10,000, the variability introduced by Monte Carlo simulations significantly influences the evaluation results, yielding a lower count of samples meeting the performance criteria. Increasing the Monte Carlo sampling iterations by a factor of ten can somewhat mitigate this problem; however, the rate of valid samples generated over time decreases.

We optimized the Monte Carlo sampling iterations to enhance the quantity of valid samples produced by VECBEE per time unit, maintaining the error allowance within 30% of the MSE limit. In the case of GPTAC, we performed generalized batch output sampling across different constraints. The numbers of valid samples produced by both techniques were statistically aggregated and compared in Table VI.

GPTAC's uniquely designed generalized generation mechanism ensures the output diversity within the specified limits. In contrast, with extensive MSE constraints, GPTAC, which has been both fine-tuned and retrained using high-quality datasets, significantly reduces sample errors. In both extreme cases, GPTAC consistently outperforms VECBEE. Specifically, in the task of exploring 8-bit signed approximate multipliers, GPTAC showed more effective exploration in over half of the assessed MSE constraints, outperforming the VECBEE synthesis tool, which utilized 32 threads at maximum capacity.

Importantly, the deployed GPTAC model demands under 4 GB of GPU memory, enabling its implementation on budget-friendly hardware for inference operations. Consequently, GPTAC serves as a computationally efficient approach for approximate circuit optimization.

VIII. CONCLUSION

This paper introduces GPTAC, a domain specific generative pre-trained model tailored for designing approximate circuits. GPTAC is capable of generating specific approximate circuits based on specified area and precision requirements by an efficient GAI approach. The improved tokenized representation facilitates feasible machine learning in the dataset. The lightweight and rapid inference of GPTAC presents a fresh viewpoint on the design of approximate circuits.

REFERENCES

[1] W. Xiao, C. Zhuo, and W. Qian, "OPACT: Optimization of approximate compressor tree for approximate multiplier," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2022, pp. 178–183.

[2] M. S. Ansari, B. F. Cockburn, and J. Han, "An improved logarithmic multiplier for energy-efficient neural computing," *IEEE Trans. Comput.*, vol. 70, no. 4, pp. 614–625, Apr. 2021.

[3] S. Nambi, U. A. Kumar, K. Radhakrishnan, M. Venkatesan, and S. E. Ahmed, "DeBAM: Decoder-based approximate multiplier for low power applications," *IEEE Embedded Syst. Lett.*, vol. 13, no. 4, pp. 174–177, Dec. 2021.

[4] S. Vahdat, M. Kamal, A. Afzali-Kusha, and M. Pedram, "TOSAM: An energy-efficient truncation- and rounding-based scalable approximate multiplier," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 5, pp. 1161–1173, Jan. 2019.

[5] S. Su et al., "VECBEE: A versatile efficiency–accuracy configurable batch error estimation method for greedy approximate logic synthesis," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 11, pp. 5085–5099, Nov. 2022.

[6] P. H. Solomon, B. Pattanaik, O. Pattanaik, D. Elamvazhudhi, and B. Shaik, "AI-driven circuit optimization for energy-efficient electronics design," in *Proc. 5th Int. Conf. Recent Trends Comput. Sci. Technol. (ICRTCST)*, Apr. 2024, pp. 56–60.

[7] J. Song et al., "CircuitVAE: Efficient and scalable latent circuit optimization," in *Proc. 61st ACM/IEEE Design Autom. Conf.*, San Francisco, CA, USA, Jun. 2024, pp. 1–6.

[8] Y. Ma, S. Roy, J. Miao, J. Chen, and B. Yu, "Cross-layer optimization for high speed adders: A Pareto driven machine learning approach," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 38, no. 12, pp. 2298–2311, Dec. 2019.

[9] H. Geng, Y. Ma, Q. Xu, J. Miao, S. Roy, and B. Yu, "High-speed adder design space exploration via graph neural processes," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 41, no. 8, pp. 2657–2670, Aug. 2022.

[10] R. Roy et al., "PrefixRL: Optimization of parallel prefix circuits using deep reinforcement learning," in *Proc. 58th ACM/IEEE Design Autom. Conf. (DAC)*, Dec. 2021, pp. 853–858.

[11] L. Chen et al., "The dawn of AI-native EDA: Opportunities and challenges of large circuit models," 2024, *arXiv:2403.07257*.

[12] N. Wu, Y. Xie, and C. Hao, "IRONMAN: GNN-assisted design space exploration in high-level synthesis via reinforcement learning," in *Proc. Great Lakes Symp. (VLSI)*, New York, NY, USA, 2021, pp. 39–44.

[13] N. Wu, Y. Xie, and C. Hao, "IronMan-pro: Multiobjective design space exploration in HLS via reinforcement learning and graph neural network-based modeling," *IEEE Trans. Comput.-Aided Design Integr. Circuits Syst.*, vol. 42, no. 3, pp. 900–913, Mar. 2023.

[14] Z. Shi et al., "DeepGate2: Functionality-aware circuit representation learning," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Oct. 2023, pp. 1–9.

[15] 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 *Proc. 43rd IEEE/ACM Int. Conf. Comput.-Aided Design*, Oct. 2024, pp. 1–9.
[16] Z. Shi et al., "DeepTPI: Test point insertion with deep reinforcement learning," in *Proc. IEEE Int. Test Conf. (ITC)*, Sep. 2022, pp. 194–203.
[17] K. Chang et al., "ChipGPT: How far are we from natural language hardware design," 2023, *arXiv:2305.14019*.
[18] Y. Fu et al., "GPT4AIGChip: Towards next-generation AI accelerator design automation via large language models," in *Proc. IEEE/ACM Int. Conf. Comput. Aided Design (ICCAD)*, Oct. 2023, pp. 1–9.
[19] X. Yao et al., "RTLRewriter: Methodologies for large models aided RTL code optimization," 2024, *arXiv:2409.11414*.
[20] C. Raffel et al., "Exploring the limits of transfer learning with a unified text-to-text transformer," *J. Mach. Learn. Res.*, vol. 21, no. 140, pp. 1–67, 2020.
[21] T. Wang et al., "What language model architecture and pretraining objective works best for zero-shot generalization," in *Proc. 39th Int. Conf. Mach. Learn.*, Jan. 2022, pp. 22964–22984.
[22] S. Bhojanapalli, C. Yun, A. S. Rawat, S. J. Reddi, and S. Kumar, "Low-rank bottleneck in multi-head attention models," in *Proc. 37th Int. Conf. Mach. Learn.*, Jan. 2020, pp. 864–873.
[23] Y. Wu et al., "A survey on approximate multiplier designs for energy efficiency: From algorithms to circuits," *ACM Trans. Design Autom. Electron. Syst.*, vol. 29, no. 1, pp. 1–37, Jan. 2024.
[24] Y. Wu, C. Chen, C. Wen, W. Qian, X. Yin, and C. Zhuo, "Approximate multiplier design for energy efficiency: From circuit to algorithm," in *Approximate Computing* (Studies in Computational Intelligence). Cham, Switzerland: Springer, 2022, pp. 51–76.
[25] M. Imani et al., "ApproxLP: Approximate multiplication with linearization and iterative error control," in *Proc. 56th Annu. Design Autom. Conf.*, Las Vegas, NV, USA, Jun. 2019, pp. 1–6.
[26] M. Imani, R. Garcia, S. Gupta, and T. Rosing, "RMAC: Runtime configurable floating point multiplier for approximate computing," in *Proc. Int. Symp. Low Power Electron. Design*, Jul. 2018, pp. 1–6.
[27] S. Narayanamoorthy, H. A. Moghaddam, Z. Liu, T. Park, and N. S. Kim, "Energy-efficient approximate multiplication for digital signal processing and classification applications," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 6, pp. 1180–1184, Jun. 2015.
[28] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in *Proc. 24th Internatioal Conf. VLSI Design*, Jan. 2011, pp. 346–351.
[29] M. Ha and S. Lee, "Multipliers with approximate 4–2 compressors and error recovery modules," *IEEE Embedded Syst. Lett.*, vol. 10, no. 1, pp. 6–9, Mar. 2018.
[30] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, "EvoApprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *Proc. Design, Autom. Test Eur. Conf. Exhib. (DATE)*, Mar. 2017, pp. 258–261.
[31] M. Češka, J. Matyáš, V. Mrázek, L. Sekanina, Z. Vašíček, and T. Vojnar, "ADAC: Automated design of approximate circuits," in *Proc. 30th Int. Conf. Comput. Aided Verification (CAV)*, Jan. 2018, pp. 612–620.
[32] V. Mrazek, M. A. Hanif, Z. Vasicek, L. Sekanina, and M. Shafique, "AutoAx: An automatic design space exploration and circuit building methodology utilizing libraries of approximate components," in *Proc. 56th ACM/IEEE Design Autom. Conf. (DAC)*, Jun. 2019, pp. 1–6.
[33] Z. Li et al., "Adaptable approximate multiplier design based on input distribution and polarity," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 30, no. 12, pp. 1813–1826, Dec. 2022.
[34] C. Meng, W. Qian, and A. Mishchenko, "ALSRAC: Approximate logic synthesis by resubstitution with approximate care set," in *Proc. 57th ACM/IEEE Design Autom. Conf. (DAC)*, Jul. 2020, pp. 1–6.

Sipei Yi (Student Member, IEEE) is currently pursuing the bachelor's degree in electronic information with the Yingcai Honors College, University of Electronic Science and Technology of China (UESTC).

He is undertaking a self-structured undergraduate curriculum combining the disciplines of communication engineering and digital integrated circuit design. His research interests include AI-assisted digital integrated circuit design and high-energy-efficiency digital circuit architectures.

Weichuan Zuo is currently pursuing the bachelor's degree in electronic information with the Yingcai Honors College, University of Electronic Science and Technology of China (UESTC).

His research interests include LLM-driven approximate circuit optimization, with a particular emphasis on employing machine learning techniques for automated design exploration and hierarchical spatial search in high-dimensional solution spaces.

Hongyi Wu is currently pursuing the master's degree in electronic information engineering with Sichuan University of Science and Engineering, China. Her research focuses on large language model-driven optimization and design of approximate circuits and neural network accelerators.

Ruicheng Dai (Member, IEEE) received the B.S. degree in microelectronics science and engineering from East China Normal University, China, in 2022. He is currently pursuing the Ph.D. degree in electronic science and technology with the University of Michigan-Shanghai Jiao Tong University Joint Institute, Shanghai Jiao Tong University, China.

His recent research focuses on developing logic synthesis techniques for approximate computing and reliability-driven circuit design.

Weikang Qian (Senior Member, IEEE) received the B.Eng. degree in automation from Tsinghua University in 2006 and the Ph.D. degree in electrical engineering from the University of Minnesota in 2011.

He is currently an Associate Professor with the University of Michigan-Shanghai Jiao Tong University Joint Institute, Shanghai Jiao Tong University. His main research interests include electronic design automation and digital design for emerging computing paradigms. His research works were nominated for the Best Paper Award from the International Conference on Computer-Aided Design (ICCAD), Design, Automation, and Test in Europe Conference (DATE), and the International Workshop on Logic and Synthesis (IWLS). He serves as an Associate Editor for IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS.

Jienan Chen (Senior Member, IEEE) received the B.S. and Ph.D. degrees in communication systems from the University of Electronic Science and Technology of China (UESTC), Chengdu, China, in 2007 and 2014, respectively.

He was with the School of Electrical and Computer Engineering, University of Minnesota, Minneapolis, MN, USA, as a Visiting Scholar, in 2012 and 2014, and then as a Post-Doctoral Scholar with the University of North Texas, Denton, TX, USA. He is currently a Professor with the National Key Laboratory of Science and Technology on Communications, UESTC. His current research interests include machine-learning-based signal processing, artificial intelligence for networking, and circuit system design.

Prof. Chen served as a TPC Member for GLOBECOM and ICC. He has received the Best Paper Award from ICCT 2022. He also served as the Track Chair for ICCT 2024 and the Symposium Chair for GlobalSIP.