AnalogToBi: Device-Level Analog Circuit Topology Generation via Bipartite Graph and Grammar Guided Decoding

Anonymous Authors[1]

Abstract

Automatic generation of device-level analog circuit topologies remains a fundamental challenge in analog design automation. Recent transformer-based approaches have shown promise, yet they often suffer from limited functional controllability, memorization of training data, and the generation of electrically invalid circuits. We propose **AnalogToBi**, a device-level analog circuit topology generation framework that addresses these limitations. AnalogToBi enables explicit functional control via a *circuit type token* and adopts a *bipartite graph-based circuit representation* that decouples positional ordering from functional semantics, encouraging structural reasoning over sequence memorization. In addition, *grammar-guided decoding* enforces electrical validity during generation, while apply *device renaming–based data augmentation*improves generalization by increasing sequence diversity without altering circuit functionality. Experimental results show that AnalogToBi achieves 97.8% validity and 92.1% novelty, resulting in 89.9% valid and novel circuits under conditional generation, without human expert involvement. We further present that generated circuits can be automatically translated into SPICE netlists, and SPICE simulations confirm that AnalogToBi discovers high-quality analog topologies that outperform prior methods.

1. Introduction

Analog circuits form the backbone of modern electronic systems, but their design remains highly dependent on expert knowledge (Liakos & Plessas, 2025). Analog circuit design typically involves first determining an appropriate circuit topology that defines the core operational properties of the circuit, followed by sizing individual circuit components (i.e., devices) to achieve optimal operating conditions within the chosen topology (Liu & Chitnis, 2025). This process requires a deep understanding of device-level behavior as well as complex inter-device interactions. As a result, analog circuit design relies heavily on skilled engineers. However, as the availability of such experienced analog designers is limited across the semiconductor industry, there is growing interest in the automation of analog circuit design.

Early work on analog design automation focused on device sizing optimization under fixed circuit topologies (Settaluri et al., 2020; Cao et al., 2022; Wang et al., 2020; Lyu et al., 2017). While these approaches can effectively refine device parameters, they suffer from a fundamental limitation: when the initial topology is poorly chosen, no amount of parameter optimization can overcome the resulting structural bottleneck to achieve the desired figure of merit (FoM). Therefore, automating circuit topology design is essential for enabling effective analog circuit design automation.

Automating circuit topology design is substantially more challenging than device sizing optimization, as it requires a thorough understanding of the complex structural and electrical relationships among devices. As a result, early efforts in analog design automation largely avoided explicit topology design, despite its importance. More recently, advances in transformer-based models, including large language models (LLMs), have renewed interest in topology design by enabling improved modeling of contextual information (Lai et al., 2025a; Chang et al., 2024; Gao et al., 2025a). Despite this renewed momentum, existing approaches for topology design still remain at an early stage and face challenges, including limited ability to generate electrically valid circuits, restricted functional controllability, or heavy reliance on the knowledge of the experienced engineers.

In this paper, to address these challenges, we propose **AnalogToBi**, a novel framework for device-level analog circuit topology generation. AnalogToBi trains a compact decoder-only Transformer model (11.3M parameters) from scratch for topology generation and is built upon four key ideas. First, a *circuit-type token* enables explicit control over the target circuit functionality during decoding. Second, a *bipartite graph-based circuit representation* separates devices

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

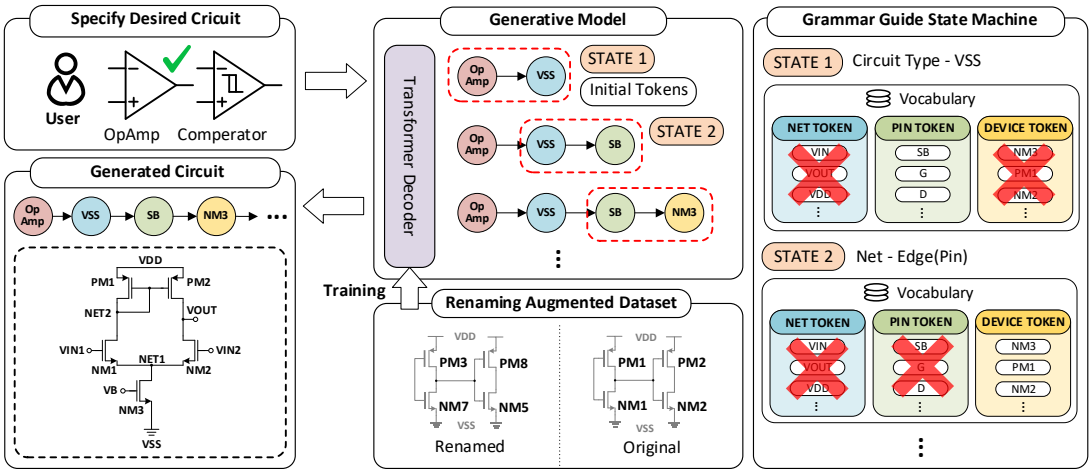

Figure 1. Overview of the proposed AnalogToBi framework. Given a user-specified circuit type, a Transformer decoder trained with renaming augmented dataset generates a device-level circuit sequence under grammar-guided decoding. The generated sequence is mapped to a bipartite device–net graph and netlist.

and nets into distinct node types, allowing compact structural descriptions while promoting generalization beyond memorized token patterns. Third, *grammar-guided decoding* restricts token generation to enforce electrical validity while preserving topological diversity. Finally, we introduce a *device renaming–based data augmentation strategy* that renames device tokens while maintaining electrical equivalence, preventing the model from simply memorizing circuit sequences. By integrating these components, AnalogToBi provides a more practical automated framework for device-level analog circuit topology generation, achieving explicit functional controllability together with high validity and novelty in the generated circuits.

2. Related Work

This section reviews prior work on the automation of analog circuit topology design. Previous approaches can be categorized into four classes: topology search, topology refinement, behavior-level topology generation, and device-level topology generation. The following subsections examine the key characteristics of each category and conclude by identifying the limitations of prior work that motivate the proposed approach.

2.1. Topology Search

Previous works in this category aim to search for a topology that meets user-specified operational goals from a predefined set of candidate topologies manually crafted based on prior analog circuit designs (Veselinovic et al., 1995; Gerlach et al., 2015; Poddar et al., 2024; Mehradfar et al., 2025). For example, Falcon (Mehradfar et al., 2025) selects the most appropriate topology given a target performance specification. As these methods are limited to selecting from predefined candidates, they struggle to generate topologies beyond those explicitly included in the predefined set.

2.2. Topology Refinement

Recent studies have explored the use of pre-trained LLMs for topology refinement, motivated by their strong capabilities in logical reasoning, code generation, and tool-integrated agentic workflows (Liu et al., 2024; Bhandari et al., 2024). Representative works in this category include AnalogCoder (Lai et al., 2025a;b), Artisan (Chen et al., 2024), AnalogExpert (Zhang et al., 2025), and AnalogSeeker (Chen et al., 2025). In these approaches, an initial circuit topology is provided by the user. Analog circuits are typically represented in a code-like format, such as SPICE netlists, to enable pre-trained LLMs to interpret circuit structures. To further enhance domain understanding, AnalogExpert (Zhang et al., 2025) introduces fine-tuning of pre-trained LLMs for analog design tasks. The LLMs are then employed as agents to iteratively refine the given topology by evaluating design quality using circuit simulation tools. While these approaches are effective in refining existing designs, the final design quality is strongly dependent on the initial topology. This reliance on expert-provided starting points restricts their ability to systematically explore novel circuit structures.

2.3. Behavioral-Level Topology Generation

Early topology generation approaches primarily adopted behavior-level topology generation strategies (Zhao & Zhang, 2019; Lu et al., 2022; Dong et al., 2023). In these approaches, key analog circuit components are abstracted as behavior-level building blocks whose types and internal de-

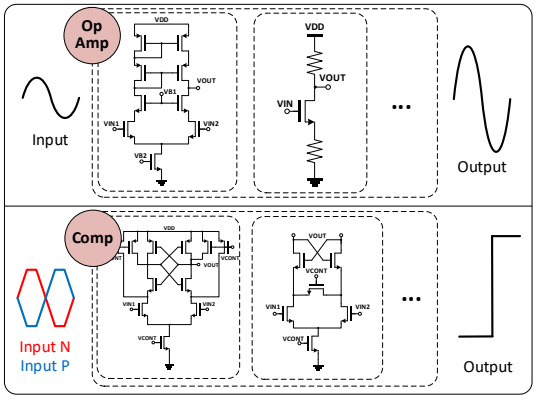

Figure 2. Example of analog circuit categorization based on functionality. OpAmps amplify input signals, while comparators (Comp) compare two input signals and generate outputs based on the comparison result. Each category is associated with a corresponding circuit type token.

signs are predefined. Topology generation is then performed by composing and connecting these abstract blocks. For example, CktGNN (Dong et al., 2023) treats single-stage operational amplifiers (OpAmps), along with resistors and capacitors, as the basic building blocks for topology generation. While this formulation enables the generation of diverse circuit structures by composing these basic units, it inherently limits design flexibility, as the internal device-level realizations of single-stage OpAmps are fixed and circuits that cannot be expressed as compositions of abstract blocks cannot be generated. As a result, behavior-level topology generation methods are unable to sufficiently capture the rich design space of device-level circuit topologies.

2.4. Device-Level Topology Generation

More recently, active research has explored device-level topology generation to enable fully flexible analog circuit design (Chang et al., 2024; 2025; Gao et al., 2025a;b; Li et al., 2025). These approaches leverage transformer-based models to directly generate device-level circuit representations. However, because the sequential representations of analog circuits differ substantially from natural language, such methods typically train dedicated models from scratch rather than relying on pre-trained language models. LaMAGIC (Chang et al., 2024; 2025), a pioneering work in this category, proposes training a new model for device-level topology generation. While effective within its target domain, LaMAGIC is limited to generating a single circuit type, namely power converter.

AnalogGenie (Gao et al., 2025a;b) demonstrates that a single model can generate multiple types of analog circuits at the device level. However, AnalogGenie lacks explicit control over circuit functionality, which limits its applicability in practical analog design scenarios. Moreover, to train a high-quality topology generation model that produces a high ratio of electrically valid circuits, AnalogGenie relies on human-in-the-loop reinforcement learning, where expert designers evaluate generated circuits and provide feedback during training. In this process, the criteria for quality evaluation are not explicitly defined and are largely dependent on individual expert judgment, thereby limiting reproducibility and scalability.

In summary, enabling fully flexible analog circuit topology design ultimately requires device-level topology generation. To overcome the limitations of existing device-level approaches and make them practical, there is a clear need for a topology generation framework that can both generate diverse circuit types with controllable functionality and support a fully automated training procedure.

3. Proposed Method

We propose AnalogToBi, a novel device-level analog circuit topology generation framework. Similar to recent approaches (Gao et al., 2025a;b), AnalogToBi employs a transformer-based model dedicated to analog circuit topology generation. To overcome the limitations of prior work, AnalogToBi introduces four key design elements, which are described in detail in the following subsections.

3.1. Circuit Type Token for Functionality Control

A model dedicated to analog circuit topology generation requires a task-specific vocabulary tailored to circuit representations. Accordingly, prior approaches (Gao et al., 2025a;b) introduce a dedicated token vocabulary composed of circuit devices and their associated pins. Since constructing such a dedicated model already involves designing a task-specific vocabulary and embedding layers from scratch, it is straightforward to incorporate additional tokens for functionality control during model construction. As illustrated in Fig. 2, analog circuits can be clearly categorized according to their functionality. For example, circuits that amplify input signals are referred to as OpAmps, while circuits that compare two input signals and produce an output based on the comparison are referred to as comparators. Within each functional category, multiple circuit topologies may exist. We refer to this functional categorization as the circuit type, and introduce a corresponding circuit type token for each category. In this work, we categorize circuits into 15 types, including 14 function-specific categories and one general category representing circuits without a predefined functionality. Each circuit type is assigned a unique token. Unlike prior approaches that initiate generation from a fixed reference token (e.g., VSS), AnalogToBi receives a circuit type token from the user and starts the generation process conditioned on this token, thereby enabling explicit control over the functionality of the generated circuit.

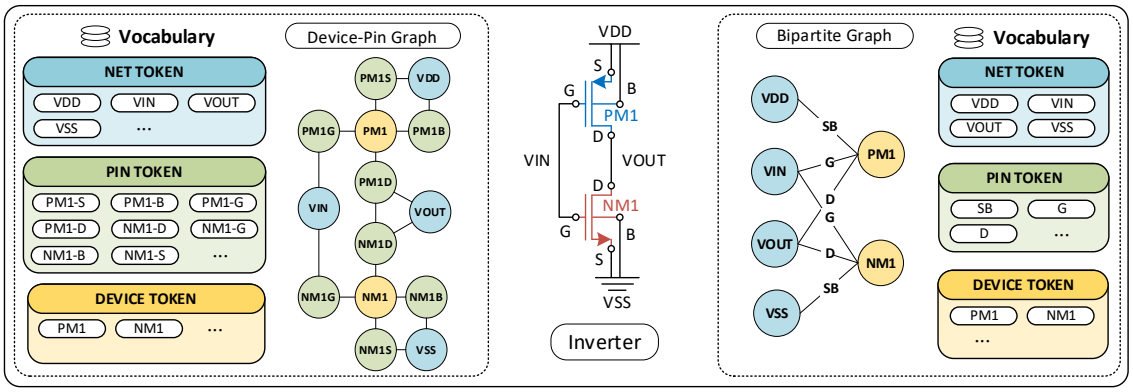

Figure 3. Comparison between conventional device–pin level representation (Gao et al., 2025a) and the proposed bipartite graph representation. The device–pin representation (left) treats each device pin as a separate token, while the bipartite representation (right) models devices and nets as distinct node types with pin semantics encoded as typed edges.

However, simply applying circuit-type tokens to existing approaches leads to increased memorization of the training data. Specifically, when circuit type tokens are introduced into the AnalogGenie framework, the model increasingly reproduces training circuit patterns rather than exploring novel topologies. This behavior is reflected in a sharp drop in the novelty ratio, defined as the fraction of generated circuits that are not present in the training dataset, which decreases from 68.0% to 45.2% after incorporating the circuit type token. These results indicate that naïvely adding circuit type conditioning encourages the model to associate functionality with surface-level token patterns, leading to memorization instead of learning transferable structural principles. This issue stems from the token design used in AnalogGenie, where individual tokens simultaneously encode both structural location information and functional semantics of circuit components. Therefore, effectively leveraging circuit-type tokens necessitates a new circuit representation strategy that decouples structural relationships from token identities and mitigates memorization.

3.2. Bipartite Graph Circuit Representation

AnalogGenie adopts a device–pin level graph representation for analog circuit modeling (Fig. 3-left), in which each pin of a device is assigned a distinct token. Since AnalogGenie supports circuit generation with up to 35 NMOS (NM) devices and 35 PMOS (PM) devices, it maintains separate token sets for device instances such as NM1, NM2, ..., NM35 and PM1, ..., PM35. Each device contains four pins, namely source (S), gate (G), drain (D), and body (B), which are used to connect to other devices or current and voltage sources. Under the device–pin level representation, a dictionary of 70 devices results in separate pin-specific token dictionaries of size 70×4, corresponding to tokens such as NM1S, PM1S, NM1G, PM1G, and so on. While this representation explicitly captures pin locations within a sequential format, it lacks representational flexibility and effectively reduces analog topology generation to an ordering problem over a fixed token set. As a result, the model is prone to memorizing token patterns observed during training rather than learning the underlying electrical relationships among circuit components. This limitation is further exacerbated under data-scarce conditions, where circuit type conditioning partitions the training data across multiple functional categories. Therefore, to effectively leverage circuit type tokens while avoiding memorization, a new circuit representation strategy is required.

To decouple positional information from functional semantics in sequential circuit representations, we explicitly leverage the inherent structural properties of analog circuits, namely devices, nets, and pins (Fig. 3-right). In analog circuit design, devices perform electrical functions, nets represent electrical connection points, and pins define the interfaces through which devices connect to nets. A key observation is that a device can connect to a net only through its pins, and conversely, a net is always connected to one or more devices. This mutual dependency implies that any analog circuit can be naturally expressed as a graph consisting of two disjoint entity types, devices and nets, with connectivity mediated exclusively by pins. Based on this observation, we represent each analog circuit as a bipartite graph composed of a device node set and a net node set. Edges between device nodes and net nodes correspond to pin-level connections and indicate which pin of a device is connected to which net. Importantly, pins are not treated as independent entities. Instead, they are modeled as intrinsic attributes of devices that define the type of connection to a net. This formulation eliminates the need to define device-specific pin tokens as standalone nodes. To process this bipartite representation with the proposed model, we categorize tokens into device-type tokens (e.g., NM1, PM1), net-type tokens (e.g., VIN1, NET1), and pin-type tokens (e.g., S, D). The bipartite graph is then serialized into a sequence using a graph traversal strategy. In this represen-

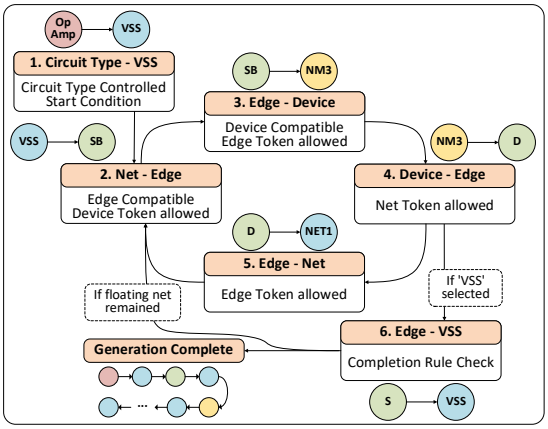

Figure 4. State machine for Grammar-guided decoding. The grammar restricts valid token transitions during circuit generation.

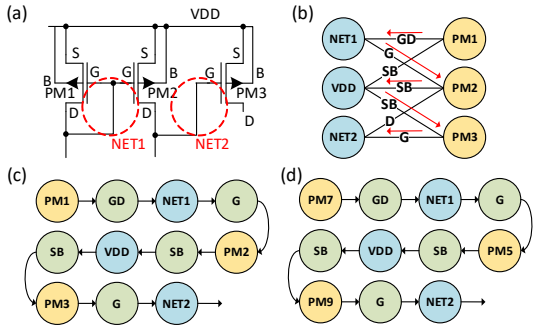

Figure 5. Overview of sequence generation and device renaming augmentation for analog circuits. (a) Original device-level circuit topology. (b) Corresponding bipartite graph representation. (c) Token sequence serialized by graph traversal. (d) Functionally equivalent sequence obtained from device renaming augmentation.

tation, the position and function of each pin are resolved relationally through adjacent device and net tokens. This contextual interpretation of pin semantics encourages the model to reason over structural context, since identifying the role of a pin requires understanding its neighboring tokens and their relationships.

However, the proposed bipartite circuit representation imposes a specific ordering constraint during token generation, in which pin-type tokens must appear between device-type and net-type tokens. If the model fails to internalize this constraint, electrically invalid configurations such as floating nodes or short circuits may arise during generation. Given the limited availability of device-level analog circuit data, it is challenging for the model to reliably learn and enforce this ordering rule through training alone. This limitation motivates the grammar-guided decoding strategy introduced in the next section.

3.3. Grammar-Guided Decoding

To enable effective topology generation based on the bipartite graph representation under limited data availability, we introduce grammar-guided decoding. Following the paradigm of grammar-constrained decoding in LLMs (Geng et al., 2023; Raspanti et al., 2025), our approach enforces circuit-level electrical and structural constraints directly during the generation of token sequences, as illustrated in Fig. 4.

We formalize the structural regularities induced by the bipartite graph circuit representation as an explicit grammar. Circuit generation is modeled as a state-transition process, where each state is defined by the combination of the token generated at the current decoding step T and the token generated at the previous step $T-1$. Rather than operating on individual token identities, states are defined in terms of token categories, namely circuit-type tokens, device-type tokens, net-type tokens, and pin-type tokens. Circuit-type tokens are provided by the user to specify the target circuit functionality and are used solely to initialize the generation process. These tokens are not generated during decoding and are therefore masked out for all subsequent generation steps.

Given the current state, the grammar restricts the set of admissible next tokens by masking invalid transitions, enforcing a structured device–pin-net transition pattern. For example, if a device-type token is generated at step T, then at step $T+1$ all tokens except pin-type tokens are masked, forcing the model to generate a pin-type token. By excluding invalid transitions from the generation trajectory, this state-machine-based decoding prevents electrically invalid configurations from being generated. Finally, sequence termination is permitted only when all device pins are fully connected and no internal net remains floating, thereby guaranteeing the electrical completeness of the generated circuit. More details on the specific grammar adopted in AnalogToBi are provided in Appendix C.

3.4. Data Augmentation with Device Renaming

Transformer-based circuit generation models operate on sequential representations, requiring each circuit to be converted into a token sequence. As illustrated in Fig. 5, each circuit (Fig. 5(a)) is first transformed into a graph representation (Fig. 5(b)) and then serialized into a sequence (Fig. 5(c)). Because different graph traversal orders produce different sequences for the same circuit graph, previous work (Gao et al., 2025a;b) introduces a data augmentation strategy that generates multiple sequence variants by applying different traversal orderings. Building on this approach, we introduce an additional device renaming augmentation that increases data diversity while preserving circuit functionality. This augmentation exploits the fact that device identifiers encode instance names rather than functional be-

havior. For example, a device token such as PM1 in Fig. 5(c) can be renamed to PM7 in Fig. 5(d) without altering the electrical functionality or connectivity of the circuit. By randomly renaming device identifiers, this augmentation substantially expands the effective training dataset while maintaining electrical equivalence among circuits.

Our experiments show that this strategy is particularly effective when combined with the proposed bipartite graph-based circuit representation in AnalogToBi. Specifically, device renaming reduces memorization of exact token sequences and encourages the model to learn structural and relational patterns, resulting in improved novelty without sacrificing validity. In contrast, when applied to the conventional device–pin level representation, device renaming results in a significant degradation of valid circuit generation. This behavior arises because the device–pin level representation tightly couple positional ordering with functional semantics, causing the model to rely on memorized token patterns rather than learning generalized structural relationships. A detailed quantitative analysis of this effect is presented in Section 4.3.

4. Experiments

4.1. Experimental Setup

Dataset. We evaluate the proposed AnalogToBi using the publicly released dataset of 2,165 analog circuits represented as SPICE-style netlists (Gao et al., 2025a). To enable circuit-type–conditioned model training, we manually categorize the dataset according to circuit functionality, resulting in 14 distinct circuit types, along with an additional generic category for circuits with undefined functionality. For each circuit category, 90% of the circuits are used for training, while the remaining 10% are reserved for validation. Each circuit is first converted into a graph representation and subsequently serialized into a sequence. We apply two forms of data augmentation: (1) the sequence augmentation strategy proposed in AnalogGenie, which generates multiple sequence variants for each circuit using different graph traversal orderings, and (2) the proposed device renaming augmentation, which renames device tokens without altering circuit functionality, thereby increasing data diversity while preserving electrical equivalence. As a result of these augmentations, the effective training dataset is expanded from 2,165 to 397,515 sequences.

Baselines. We compare AnalogToBi with recent approaches for analog circuit generation, including AnalogCoder(-Pro) (Lai et al., 2025a;b), LaMAGIC (Chang et al., 2024), and AnalogGenie(-Lite) (Gao et al., 2025a;b). These methods differ from AnalogToBi in terms of circuit representation, generation capability, and scalability, as discussed in Section 2. AnalogCoder leverages foundation large language models for circuit topology refinement, while LaMAGIC focuses on generating power converter circuits within a single circuit domain. As AnalogGenie supports the generation of multiple circuit types but does not provide explicit control over circuit functionality, it serves as the strongest baseline for AnalogToBi. For all baselines, we follow the original implementations and experimental settings to reproduce the reported results.

Implementation Details. To ensure a fair comparison with the strong baseline AnalogGenie, we align the model architecture used for AnalogToBi, thereby isolating the impact of the proposed circuit representation and grammar-guided decoding mechanisms. The decoder-only transformer has an embedding dimension of 384, 6 attention heads, and 6 transformer layers, resulting in approximately 11.3M parameters. Note that the model size of AnalogToBi is slightly smaller than that of AnalogGenie (11.8M parameters), due to the more compact token vocabulary and corresponding embedding layers enabled by the proposed bipartite representation, as discussed in Section 3.2. The maximum context length is fixed to 1024 tokens, and a dropout rate of 0.2 is applied throughout the network. Training is performed using AdamW with a learning rate of 3×10^{-4} and a batch size of 64 for 100,000 iterations. The entire training process takes 22.8 hours on an NVIDIA RTX 5880 GPU. During inference, circuits are generated autoregressively using multinomial sampling with a fixed temperature of 0.7.

Metrics. To properly evaluate the ability of the circuit generation model to generate structurally correct and novel circuit topologies, we assess performance using three metrics: Validity, Novelty, and Valid & Novel. **Validity** measures the fraction of generated circuits that are electrically valid, such that they can be converted into SPICE netlists without errors. **Novelty** measures the fraction of generated circuits that are novel. A circuit is considered novel if its topology is not isomorphic to any circuit in the training set. Finally, **Valid & Novel** measures the proportion of generated circuits that are both electrically valid and novel.

Circuit Type Classification. To quantitatively evaluate whether generated circuits reflect the intended functionality, we employ a Graph Attention Network (GAT) to classify the circuit types of generated circuits. Architectural and training details of the classifier are provided in Appendix B. The GAT achieves a classification accuracy of 99.91%, indicating high reliability. Accordingly, for each circuit-type–conditioned generation task, we measure the type classification accuracy using the GAT classifier.

4.2. Main Results

Table 1 provides a comprehensive comparison between AnalogToBi and baseline methods in terms of both method properties and generation quality. While AnalogCoder(-pro)

Table 1. Comparison of AnalogToBi with prior topology design methods. The Properties columns summarize the key characteristics of each method. *Train* indicates the training strategy used for circuit design (FT: fine-tuning; Full: training from scratch). *HITL* (Hunman-in-the-Loop) denotes whether human experts are involved in the training or inference process. *# Type* represents the number of circuit types supported by each method, and *Type Ctrl* indicates whether explicit control over the generated circuit type is provided. All evaluation results are taken from the original papers. Results for AnalogGenie* are reproduced using the officially released source code.

	Properties						Evaluation Results		
Method	Model (# Param.)	Train	HITL	Mode	# Type	Type Ctrl	Validity	Novelty	Val.&Nov.
AnalogCoder (Lai et al., 2025a)	GPT-4o (–)	×	○	Refinement	24	○	66.1%	–	–
AnalogCoder-pro (Lai et al., 2025b)	GPT-5 (–)	×	○	Refinement	28	○	91.1%	–	–
LaMAGIC (Chang et al., 2025)	Flan-T5 (250M)	FT	×	Generation	1	×	96.0%	–	–
AnalogGenie (Gao et al., 2025a)	Custom (11.8M)	Full	×	Generation	11	×	73.5%	98.9%	–
AnalogGenie (Gao et al., 2025a)	Custom (11.8M)	Full	○	Generation	11	×	93.2%	99.0%	–
AnalogGenie-Lite (Gao et al., 2025b)	Custom (11.8M)	Full	○	Generation	11	×	97.0%	99.8%	–
AnalogGenie* (Gao et al., 2025a)	Custom (11.8M)	Full	×	Generation	11	×	58.2%	68.0%	26.2%
AnalogToBi (Proposed)	**Custom (11.3M)**	**Full**	×	**Generation**	**15**	○	**97.8%**	**92.1%**	**89.9%**

Table 2. Breakdown of AnalogToBi generation results by circuit type. (OpAmp: operational amplifier; Mirror: current mirror; Comp: comparator; Mix: mixer; LDO: low-dropout regulator; Oscil: oscillator; Filt: filter; BGR: bandgap reference; PAmp: power amplifier; VoltR: voltage regulator; PConv: power converter; PLL: phase-locked loop; S Cap: switched-capacitor; A/D_D/A: data converter)

	OpAmp	Mirror	Comp	Mix	LDO	Oscil	Filt	BGR	PAmp	VoltR	PConv	PLL	S Cap	A/D_D/A	General	Avg.
Validity (%)	97.6	99.4	97.1	99.3	95.6	97.7	98.9	97.7	97.7	98.5	96.2	97.8	95.9	99.6	98.1	**97.8**
Novelty (%)	89.1	77.4	92.0	84.1	97.6	92.1	92.9	94.0	99.7	91.0	99.2	90.9	91.4	96.5	92.9	**92.1**
Valid & Novel (%)	86.7	76.8	89.1	83.5	93.2	89.8	91.8	91.7	97.4	89.5	95.4	88.7	87.3	96.1	91.0	**89.9**
Type acc. (%)	99.3	89.0	70.8	99.8	100.0	96.4	92.0	99.9	91.3	85.1	100.0	92.9	68.8	93.1	–	**91.3**

support multiple circuit types with explicit type control, they rely on large-scale foundation models such as GPT-5 and operate only in a topology refinement setting, where a base topology design must be provided as input prompt. As a result, these methods do not address the problem of generating novel circuit topologies from scratch. Among topology generation methods, LaMAGIC demonstrates high validity (96.0%) but is restricted to a single circuit domain, namely power converters. In contrast, AnalogGenie(-Lite) supports multi-type topology generation but does not provide explicit control over the generated circuit type. Moreover, without human-in-the-loop (HITL) engagement during training, AnalogGenie achieves only 73.5% validity, which further drops to 58.2% when reproduced using the officially released source code.

AnalogToBi addresses these limitations by jointly leveraging circuit-type conditioning, bipartite graph representation, and grammar-guided decoding. AnalogToBi achieves a Validity of 97.8%, comparable to or exceeding all prior generation-based methods, while maintaining a high Novelty of 92.1%. More importantly, the Valid & Novel ratio of 89.9% indicates that AnalogToBi consistently generates circuits that are both electrically valid and structurally novel. Table 2 further breaks down the generation results of AnalogToBi by circuit type after generating 1,000 circuits per type. On average, the accuracy of generating the desired circuit type is 91.3%, demonstrating strong functional controllability. For Comparator, Voltage Regulator, and Switched-Capacitor circuits, the classification accuracy is much lower than the average, which can be attributed to the limited number of training samples in these categories (Fewer than 10 circuits per type. See Appendix I for the detailed dataset distribution.) Nevertheless, AnalogToBi is still able to learn meaningful structural patterns for these underrepresented categories through the proposed circuit representation and training strategy. Taken together, these results position AnalogToBi as a practical foundation for fully automated, device-level analog circuit topology design.

4.3. Ablation Study

Table 3 presents an ablation study of the proposed AnalogToBi framework to examine the impact of key design components. Evaluation metrics include Validity, Novelty, Valid & Novel, circuit type classification accuracy, and 10-gram matching rate. The 10-gram matching rate measures the fraction of generated circuits whose sequences contain an exact match of 10 consecutive tokens appearing at the beginning and the end of sequences in the training dataset, and is used as an indicator of memorization.

Even without circuit-type tokens or device renaming augmentation, the bipartite graph representation achieves higher validity and substantially lower 10-gram matching rates compared to the conventional device–pin representation. This indicates that the bipartite representation inherently reduces the tendency to memorize training circuits. When circuit-type tokens are introduced, both representations exhibit improved validity, as the target circuit patterns become more constrained. However, because circuit-type condition-

Table 3. Ablation study of the proposed AnalogToBi to evaluate the impact of key design components, including the circuit-type token, circuit representation, and device renaming. Grammar-guided decoding is inherently coupled with the bipartite graph representation.

Options			Evaluation Results				
Representation	Type Token	Rename	Validity ↑	Novelty ↑	Val.&Nov. ↑	Type Acc. ↑	10-Gram Match ↓
Device-pin	×	×	58.2%	68.0%	26.2%	–	59.7%
Device-pin	○	×	75.2%	45.2%	20.4%	97.9%	73.6%
Device-pin	○	○	45.0%	90.0%	35.1%	89.9%	1.7%
Bipartite	×	×	61.8%	57.3%	19.9%	–	24.6%
Bipartite	○	×	97.8%	74.6%	72.4%	92.5%	16.6%
Bipartite	○	○	**97.8%**	**92.1%**	**89.9%**	**91.3%**	**0.0%**

ing partitions the dataset into smaller subsets, memorization becomes more severe for the device–pin representation, whereas the bipartite representation remains robust against this effect. Finally, when device renaming data augmentation is applied, the limitations of the device–pin representation become more pronounced. Since device–pin tokens tightly couple positional and functional information, renaming disrupts memorized token patterns and leads to a significant drop in validity. In contrast, the proposed bipartite representation preserves high validity while further improving novelty, demonstrating its ability to generalize structural relationships rather than relying on surface-level token memorization.

4.4. Case Study with SPICE Simulation

We further conduct case studies on generated circuit topologies using SPICE simulations with the LTspice under 180nm CMOS process. To enable direct evaluation using standard analog simulation workflows, we implement an automated sequence-to-SPICE translation pipeline that converts generated sequences into executable SPICE netlists. During translation, a simple rule-based device sizing procedure is applied. (See Appendix G for details)

Figure 6 presents an example operational amplifier generated by AnalogToBi. The circuit is evaluated under a capacitive load of 100 pF using transient and AC analyses. The generated OpAmp achieves a FoM (i.e., MHz · pF/mW) of 168.5, substantially outperforming reported results from AnalogCoder (FoM 1.7) and AnalogGenie (FoM 36.5). Despite relying only on rule-based sizing, the circuit exhibits a moderate DC gain of 29.9,dB and a wide bandwidth (BW 2.38 MHz, GBW 74.1 MHz) which is better then AnalogGenie (GBW 12 MHz), demonstrating that AnalogToBi can discover practically competitive analog topologies.

Notably, this performance is achieved without computationally expensive sizing optimization, whereas prior works typically rely on more complex device sizing optimization procedures. We attribute the superior FoM of AnalogToBi to its ability to generate a diverse set of electrically valid device-level topologies. As discussed in the previous section, unlike approaches that focus on topology refinement or are prone to memorization, AnalogToBi enables broader exploration of the design space. These results highlight that flexibility in topology generation plays a key role in achieving high analog performance, underscoring the practical value of AnalogToBi for automated device-level analog circuit design. Additional case studies are provided in Appendix E.

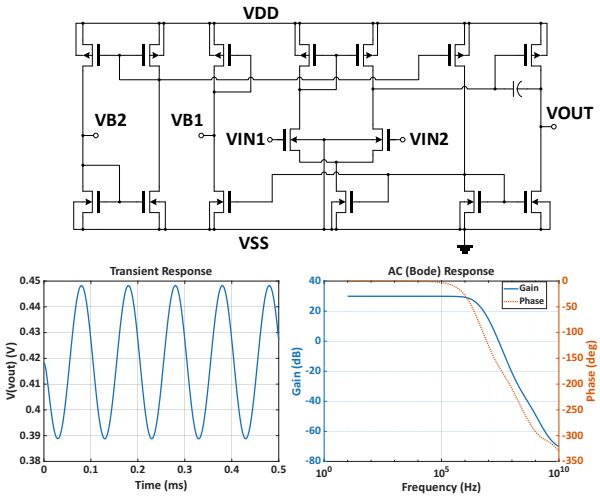

Figure 6. OpAmp topology generated with AnalogToBi and its simulation results. (Top) device-level topology, (bottom) transient response with 1mV sine, 10kHz frequency input, and AC response.

5. Conclusion

In this work, we propose AnalogToBi, a novel framework for device-level analog circuit topology generation. To enable explicit circuit-type control while generating electrically valid and structurally novel topologies without relying on predefined templates or human-in-the-loop intervention, AnalogToBi integrates circuit-type tokens, a bipartite graph representation with grammar-guided decoding, and device renaming data augmentation. Experimental results demonstrate that AnalogToBi consistently outperforms existing methods in generating valid and novel circuit topologies. Overall, AnalogToBi provides a practical foundation for fully automated, device-level analog circuit design.

Impact Statement

This work focuses on automating device-level analog circuit topology generation to improve design productivity and exploration efficiency. The primary impact of this research is expected to be positive, by reducing manual effort in analog circuit design and enabling faster prototyping of diverse circuit topologies. The proposed method operates within existing electronic design automation workflows and does not introduce new risks related to data privacy, surveillance, or autonomous decision-making. While automated circuit generation tools could potentially lower barriers to hardware development, their use remains constrained by fabrication costs, verification requirements, and domain expertise. Overall, we do not foresee significant negative societal impacts arising directly from this work.

References

Bhandari, J., Bhat, V., He, Y., Rahmani, H., Garg, S., and Karri, R. Masala-chai: A large-scale spice netlist dataset for analog circuits by harnessing ai. *arXiv preprint arXiv:2411.14299*, 2024.

Cao, W., Benosman, M., Zhang, X., and Ma, R. Domain knowledge-infused deep learning for automated analog/rf circuit parameter optimization. In *Proceedings of the 59th ACM/IEEE Design Automation Conference (DAC)*, pp. 1015–1020, 2022.

Chang, C. C., Shen, Y., Fan, S., Li, J., Zhang, S., Cao, N., and Zhang, X. Lamagic: Language-model-based topology generation for analog integrated circuits. *arXiv preprint arXiv:2407.18269*, 2024.

Chang, C. C., Lin, W. H., Shen, Y., Chen, Y., and Zhang, X. Lamagic2: Advanced circuit formulations for language model-based analog topology generation. *arXiv preprint arXiv:2506.10235*, 2025.

Chen, Z., Huang, J., Liu, Y., Yang, F., Shang, L., Zhou, D., and Zeng, X. Artisan: Automated operational amplifier design via domain-specific large language model. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, 2024.

Chen, Z., Zhuang, J., Shen, J., Ke, X., Yang, X., Zhou, M., and Yang, F. Analogseeker: An open-source foundation language model for analog circuit design. *arXiv preprint arXiv:2508.10409*, 2025.

Dong, Z., Cao, W., Zhang, M., Tao, D., Chen, Y., and Zhang, X. Cktgnn: Circuit graph neural network for electronic design automation. *arXiv preprint arXiv:2308.16406*, 2023.

Gao, J., Cao, W., Yang, J., and Zhang, X. Analoggenie: A generative engine for automatic discovery of analog circuit topologies. *arXiv preprint arXiv:2503.00205*, 2025a.

Gao, J., Cao, W., and Zhang, X. Analoggenie-lite: Enhancing scalability and precision in circuit topology discovery through lightweight graph modeling. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025b.

Geng, S., Josifoski, M., Peyrard, M., and West, R. Grammar-constrained decoding for structured nlp tasks without finetuning. *arXiv preprint arXiv:2305.13971*, 2023.

Gerlach, A., Scheible, J., Rosahl, T., and Eitrich, F. T. A generic topology selection method for analog circuits demonstrated on the ota example. In *PhD Research in Microelectronics and Electronics (PRIME)*, pp. 77–80, 2015.

Lai, Y., Lee, S., Chen, G., Poddar, S., Hu, M., Pan, D. Z., and Luo, P. Analogcoder: Analog circuit design via training-free code generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 379–387, 2025a.

Lai, Y., Poddar, S., Lee, S., Chen, G., Hu, M., Yu, B., and Pan, D. Z. Analogcoder-pro: Unifying analog circuit generation and optimization via multi-modal llms. *arXiv preprint arXiv:2508.02518*, 2025b.

Li, Q., Hong, S., Gao, J., Zhang, X., Lan, T., and Cao, W. Analogfed: Federated discovery of analog circuit topologies with generative ai. *arXiv preprint arXiv:2507.15104*, 2025.

Liakos, K. G. and Plessas, F. Analog design and machine learning: A review. *Electronics*, 14(17):3541, 2025. doi: 10.3390/electronics14173541.

Liu, C. and Chitnis, D. Eesizer: LLM-based AI agent for sizing of analog and mixed-signal circuits. *arXiv preprint arXiv:2509.25510*, 2025.

Liu, C., Liu, Y., Du, Y., and Du, L. Ladac: Large language model-driven auto-designer for analog circuits. *Authorea Preprints*, 2024.

Lu, J., Lei, L., Yang, F., Shang, L., and Zeng, X. Topology optimization of operational amplifier in continuous space via graph embedding. In *Design, Automation & Test in Europe (DATE)*, pp. 142–147, 2022.

Lyu, W., Xue, P., Yang, F., Yan, C., Hong, Z., Zeng, X., and Zhou, D. An efficient bayesian optimization approach for automated optimization of analog circuits. *IEEE Transactions on Circuits and Systems I*, 65(6):1954–1967, 2017.

Mehradfar, A., Zhao, X., Huang, Y., Ceyani, E., Yang, Y., Han, S., and Avestimehr, S. Falcon: An ml framework for fully automated layout-constrained analog circuit design. *arXiv preprint arXiv:2505.21923*, 2025.

Poddar, S., Budak, A., Zhao, L., Hsu, C. H., Maji, S., Zhu, K., and Pan, D. Z. A data-driven analog circuit synthesizer with automatic topology selection and sizing. In *Design, Automation & Test in Europe (DATE)*, 2024.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. Language models are unsupervised multitask learners. OpenAI Technical Report, 2019.

Raspanti, F., Ozcelebi, T., and Holenderski, M. Grammar-constrained decoding makes large language models better logical parsers. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pp. 485–499, Vienna, Austria, 2025. Association for Computational Linguistics.

Settaluri, K., Haj-Ali, A., Huang, Q., Hakhamaneshi, K., and Nikolic, B. Autockt: Deep reinforcement learning of analog circuit designs. *arXiv preprint arXiv:2001.01808*, 2020.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., and Polosukhin, I. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2018.

Veselinovic, P., Leenaerts, D., Van Bokhoven, W., Leyn, F., Proesmans, F., Gielen, G., and Sansen, W. A flexible topology selection program as part of an analog synthesis system. In *European Design and Test Conference (EDTC)*, pp. 119–123, 1995.

Wang, H., Wang, K., Yang, J., Shen, L., Sun, N., Lee, H. S., and Han, S. Gcn-rl circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning. In *Proceedings of the 57th ACM/IEEE Design Automation Conference (DAC)*, 2020.

Zhang, H., Sun, S., Lin, Y., Wang, R., and Bian, J. Analogxpert: Automating analog topology synthesis by incorporating circuit design expertise into large language models. In *International Symposium of Electronics Design Automation (ISEDA)*, pp. 772–777, 2025.

Zhao, Z. and Zhang, L. Graph-grammar-based analog circuit topology synthesis. In *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2019.

Table 4. Token categories used in AnalogToBi. Only representative examples are shown.

Category	Example Tokens
Circuit Type Token	`CIRCUIT_OpAmp`, `CIRCUIT_LDO`, `CIRCUIT_Filter`, `CIRCUIT_Comparator`
Device Token	`NM1`, `PM2`, `NPN1`, `R1`, `C1`, `L1`, `DIO1`
Net Token	`VSS`, `VDD`, `NET1`, `NET2`, `VIN1`, `VOUT1`
Edge (Pin) Token	`M_G`, `M_S`, `M_GD`, `B_C`, `R_C`
Termination Token	`TRUNCATE`

A. Vocabulary Definition

Table 4 summarizes the major token categories used in AnalogToBi. Rather than enumerating the full vocabulary, we present representative examples for clarity.

Circuit Type Tokens specify the target circuit functionality and condition the Transformer decoder during generation. By prepending a circuit type token (e.g., `CIRCUIT_OpAmp` or `CIRCUIT_LDO`), circuit generation is formulated as a conditional sequence modeling problem.

Device Tokens represent active and passive circuit components. These include MOSFETs, BJTs, and passive devices such as resistors, capacitors, inductors, and diodes. Each device instance is uniquely indexed to preserve structural identity in the generated topology.

Net Tokens correspond to electrical connection points in the circuit graph. They represent both internal nets (e.g., `NET1`, `NET2`) and external ports (e.g., `VSS`, `VDD`, `VIN`, `VOUT`), enabling explicit modeling of circuit connectivity.

Edge (Pin) Tokens encode device pin–net connectivity as typed edges in the bipartite graph. Token prefixes indicate device types (`M_`: MOSFET, `B_`: BJT, `R_`/`C_`/`L_`/`D_`: resistor, capacitor, inductor, diode), while suffixes denote pin semantics (e.g., `G`: gate, `S`: source, `SB`: source/body, `C`: collector). This design allows pin semantics to be shared across devices while keeping the graph representation compact.

Finally, the `TRUNCATE` token is used for sequence padding after generation terminates, indicating that no further structural tokens follow.

B. Model Architecture and Training Details

B.1. GPT Decoder

We use a decoder-only Transformer language model for autoregressive circuit generation, following the standard GPT-style architecture (Vaswani et al., 2017; Radford et al., 2019).

The model uses an embedding dimension of 384 with 6 attention heads and 6 Transformer layers. The maximum context length is fixed to 1024 tokens, and a dropout rate of 0.2 is applied throughout the network. All linear and embedding layers are initialized from $\mathcal{N}(0, 0.02)$. The resulting model contains approximately 11.3M parameters.

Training is performed using AdamW with a learning rate of 3×10^{-4} and batch size 64 for 100,000 iterations. Validation is conducted every 500 iterations, and the model checkpoint achieving the lowest validation loss (0.2763 at step 96,500) is selected; the entire training process takes 22.8 hours on an NVIDIA RTX 5880.

During inference, circuits are generated autoregressively using multinomial sampling with a fixed temperature of 0.7 and a maximum sequence length of 1024. Grammar-guided decoding (Section 6) is applied as an external constraint by masking invalid token transitions. Under this setting, the average inference latency is 1.45 s per sample.

B.2. GAT Classifier

We train a Graph Attention Network (GAT) classifier (Veličković et al., 2018) to evaluate whether generated circuit topologies match the intended circuit type. The classifier is used exclusively for evaluation and does not provide feedback to the generator.

Node and edge(pin)-type tokens are embedded with dimensionality 64, and edge embeddings are linearly projected to match the edge feature dimension used in attention. The classifier consists of three stacked GAT layers, using four attention heads in the first two layers and a single attention head in the final layer. Batch normalization is applied after each GAT layer, while ELU activation and dropout with rate 0.3 are used after all but the final layer. The resulting model contains 1.08M parameters.

During training and inference, each circuit is represented as a graph $G = (V, E)$, where nodes correspond to devices or nets and edges encode connectivity types(i.e., pins) between them.

For the k-th attention head, the attention coefficient between nodes i and j is computed as

$$e_{ij}^{(k)} = \mathrm{LeakyReLU}\left(\mathbf{a}_k^\top \left[\mathbf{W}_k\mathbf{h}_i \,\|\, \mathbf{W}_k\mathbf{h}_j \,\|\, \mathbf{e}_{ij}\right]\right), \quad (1)$$

and the normalized attention coefficient is given by

$$\alpha_{ij}^{(k)} = \mathrm{softmax}_j\left(e_{ij}^{(k)}\right), \quad (2)$$

where $\mathbf{h}_i$ and $\mathbf{h}_j$ denote node embeddings and $\mathbf{e}_{ij}$ represents the corresponding edge-type embedding.

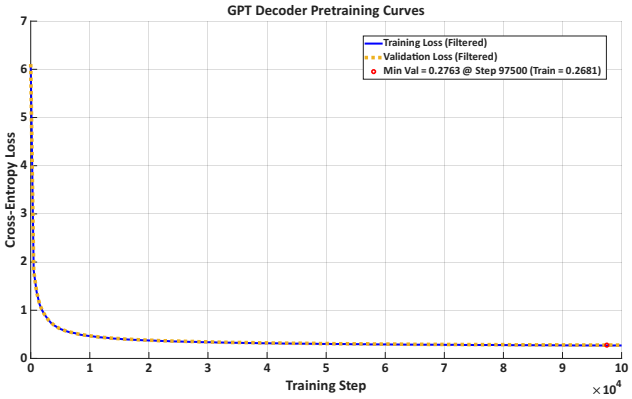

Figure 7. Training and validation loss curves for the GPT decoder during pretraining. The training and validation losses closely overlap throughout training, indicating stable convergence and the absence of overfitting.

A permutation-invariant pooling operation is applied to obtain a graph-level representation

$$\mathbf{h}_G = \mathrm{Pool}\left(\left\{\mathbf{h}_i^{(L)} \mid i \in V\right\}\right), \tag{3}$$

which serves as a graph-level embedding capturing the global circuit topology. This embedding is followed by a two-layer multilayer perceptron to predict one of 15 circuit categories.

The classifier is trained for 100 epochs using the Adam optimizer with learning rate 5×10^{-4} and weight decay 10^{-3}, with cross-entropy loss and label smoothing of 0.1. A cosine annealing learning-rate schedule and gradient clipping with maximum norm 1.0 are applied to stabilize training. The model is trained on an NVIDIA RTX 5880 GPU for 1.86 hours, achieving a Top-1 accuracy of 99.91% on the validation set, with an average inference latency of 2.02 ms per sample.

B.3. Training Curves and Accuracy

Figure 7 and Figure 8 report the training dynamics of the GPT decoder and the GAT classifier, respectively.

C. Grammar-Guided Decoding Specification

The grammar-guided decoding mechanism constrains autoregressive generation through rule-based token constraint. This appendix formalizes the induced states, transition rules, and termination conditions.

C.1. State Definitions

Decoding states are defined by the semantic roles of the most recent tokens. Let the tokens at time steps $T-1$ and T denote the two most recently generated tokens. The ordered token pair (t_{T-1}, t_T) uniquely determines the current decoding state and the set of admissible next tokens.

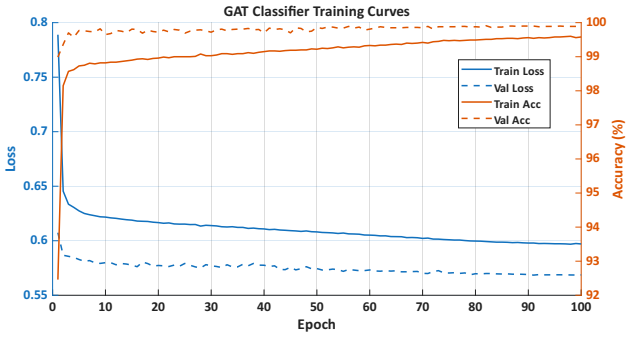

Figure 8. Training curves for the GAT-based circuit type classifier. The model achieves high validation accuracy with stable convergence behavior, demonstrating that circuit functionality can be reliably inferred from device-level topology graphs.

We define the following six states:

1. **Circuit Type – VSS.** The initial decoding state, defined by a circuit-type token followed by the reference ground net `VSS`.
2. **Net – Edge.** A state in which a net or external port token is followed by an edge-type token.
3. **Edge – Device.** A state in which an edge-type token is followed by a device token.
4. **Device – Edge.** A state in which a device token is followed by an edge-type token.
5. **Edge – Net.** A state in which an edge-type token is followed by a net or external port token.
6. **Edge – VSS.** A special case of Edge–Net in which the net token corresponds to the reference ground `VSS`.

C.2. Transition Rules

Transitions between decoding states are enforced through token masking at each time step. At time $T+1$, the set of admissible tokens is determined by the semantic types of the tokens generated at time steps $T-1$ and T, together with auxiliary decoding buffers that track partial connectivity.

Decoding buffers. The following buffers are maintained during decoding:

- A two-token history buffer storing (t_{T-1}, t_T).
- A device–edge association buffer recording previously established connections between devices, edge types, and nets.
- A net visitation buffer tracking nets encountered during the traversal.

- A device context buffer storing the most recently generated device token.

State transitions.

- **Circuit Type – VSS → Net – Edge.**
  This transition corresponds to the case where the token at time $T-1$ is a circuit-type token and the token at time T is the reference ground net `VSS`. At time $T+1$, the decoder permits edge-type tokens to initiate the traversal of the bipartite graph from the reference node.

- **Net – Edge → Edge – Device.**
  This transition corresponds to the case where the token at time $T-1$ is a net or external port (e.g., `VSS`, `VIN`, `NET*`) and the token at time T is an edge-type token (e.g., `M_S`, `M_D`). Under this state, the token at time $T+1$ must be a device token compatible with the semantics of the edge type.

- **Edge – Device → Device – Edge.**
  This transition corresponds to the case where the token at time $T-1$ is an edge-type token and the token at time T is a device token. At time $T+1$, the decoder permits edge-type tokens associated with the device category of t_T, enforcing type-level consistency.

- **Device – Edge → Edge – Net.**
  This transition corresponds to the case where the token at time $T-1$ is a device token and the token at time T is an edge-type token. At time $T+1$, the decoder restricts the admissible tokens to net or external port tokens, while recording the resulting device–edge–net association.

- **Edge – Net → Net – Edge.**
  This transition corresponds to the case where the token at time $T-1$ is an edge-type token and the token at time T is a net or external port token. Decoding proceeds by generating another edge-type token incident to the current net.

- **Edge – Net → Edge – VSS.**
  This transition is a special case of Edge–Net where the token at time T is `VSS`. At time $T+1$, the decoder permits edge-type tokens as usual; additionally, the `TRUNCATE` token becomes admissible only if the electrical rule checks (e.g., complete pin usage and internal-net connectivity) are satisfied.

At each decoding step, tokens that violate the above transition rules are masked out and assigned zero probability.

C.3. Termination Conditions

Decoding terminates when one of the following conditions is satisfied:

1. A `TRUNCATE` token is generated in the Edge–Net state with `VSS` at time T, and structural constraints are satisfied, including complete assignment of device terminals and valid connectivity of all internal nets.
2. The maximum allowed sequence length is reached.
3. No valid tokens remain after masking, indicating that decoding has reached a dead-end under the grammar constraints (e.g., due to terminal usage or reconnection restrictions), and the sequence is marked as invalid.

These termination conditions ensure that decoding halts only after the bipartite traversal returns to the reference ground node `VSS` and essential structural requirements are fulfilled.

D. Data Augmentation Algorithm

We summarize the structure-preserving data augmentation procedure used in this work in Algorithm 1. The algorithm generates multiple distinct token sequences from a single bipartite circuit graph by varying traversal orders while preserving exact topological structure. All augmented sequences are required to cover every device and net, satisfy electrical rule checks, and remain compatible with grammar-guided decoding.

Algorithm 1 Structure-Preserving Data Augmentation for Bipartite Circuit Sequences

Input: Bipartite circuit graph $G = (V, E)$ with typed edges; reference node $v_{\text{ref}} = \texttt{VSS}$; maximum augmented sequences K; maximum sequence length $T_{\max} = 1024$.
Output: Augmented sequence set $\mathcal{S}$.
Initialize $\mathcal{S} \leftarrow \emptyset$.
for $k = 1$ **to** K **do**
  Initialize $s \leftarrow [\,]$, $v \leftarrow v_{\text{ref}}$.
  Initialize visited edge set $\mathcal{E}_{\text{vis}} \leftarrow \emptyset$ and visited node set $\mathcal{V}_{\text{vis}} \leftarrow \{v_{\text{ref}}\}$.
  Append v to s.
  Randomly shuffle neighbor orderings for all nodes in G.
  while $\mathcal{E}_{\text{vis}} \neq E$ **do**
    Choose next step using the following priority:
      (i) an unvisited incident edge $e = (v, u)$;
      (ii) an edge leading to an unvisited node u;
      (iii) a shortest path from v to any node incident to an unvisited edge.
    Let the chosen move be $v \to e \to u$ (allow revisits if needed for coverage).
    Append edge-type token $\tau(e)$ and node token u to s.
    Update $\mathcal{E}_{\text{vis}} \leftarrow \mathcal{E}_{\text{vis}} \cup \{e\}$ and $\mathcal{V}_{\text{vis}} \leftarrow \mathcal{V}_{\text{vis}} \cup \{u\}$.
    Set $v \leftarrow u$.
  end while
  Close the loop: append a path from current v back to v_{ref} (if $v \neq v_{\text{ref}}$), appending corresponding $\tau(e)$ and node tokens along the path.
  Coverage check: if $\mathcal{V}_{\text{vis}} \neq V$, discard this sequence and **continue**.
  ERC check: if $\text{ERC}(s)$ fails (e.g., floating internal nets or incomplete device connections), discard this sequence and **continue**.
  Pad s with `TRUNCATE` to obtain fixed length $T_{\max}$.
  Add s to $\mathcal{S}$.
end for
return $\mathcal{S}$.

Here, $G = (V, E)$ denotes a bipartite circuit graph with device and net nodes, and $\tau(e)$ denotes the discrete edge-type token associated with edge e, encoding pin or connection semantics. $\mathcal{V}_{\text{vis}}$ and $\mathcal{E}_{\text{vis}}$ represent the sets of visited nodes and edges during sequence construction, respectively. K controls the maximum number of augmented sequences generated per circuit, and $T_{\max}$ denotes the fixed maximum sequence length after padding. $\text{ERC}(s)$ denotes an electrical rule check that filters out sequences with invalid connectivity, such as floating internal nets or incomplete device connections.

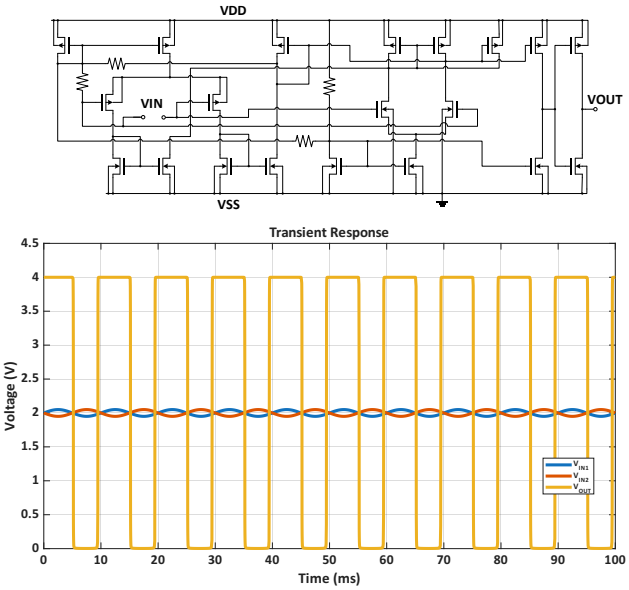

Figure 9. Transient response of the generated comparator under a 50 mV differential sinusoidal input. The output exhibits rail-to-rail swing fully from `VDD` to `VSS`.

E. Additional Case Study

E.1. Comparator Case Study with Transient Simulation

We present a case study of a generated comparator topology to demonstrate the functional validity of the proposed generation framework. Figure 9 (top) shows the generated transistor-level comparator circuit. The topology comprises a differential input stage, a regenerative amplification path, and a pull-up/pull-down output stage, forming a structure consistent with dynamic comparator architectures.

To evaluate its behavior, the generated sequence is automatically translated into a SPICE netlist using the proposed sequence-to-SPICE pipeline and simulated under a 180 nm CMOS process. A simple rule-based sizing strategy is applied to all devices without any iterative sizing optimization. The comparator is driven by a $50\,\text{mV}$ differential sinusoidal input centered around the common-mode voltage.

Figure 9 (bottom) presents the corresponding transient response obtained from SPICE simulation. Despite the small

differential input amplitude, the output exhibits clear regenerative behavior and switches decisively between the supply rails. Specifically, the output node transitions fully from VDD to VSS, demonstrating robust rail-to-rail swing and confirming correct comparator functionality.

These results indicate that the proposed framework not only guarantees structural validity at the transistor level, but also synthesizes well-composed comparator architectures in which the signal path from the preamplification stage to the inverter-based output stage is largely well-formed. In addition, the generated topology exhibits coherent biasing across stages, enabling stable operation under realistic analog simulation conditions, even when relying solely on rule-based device sizing.

F. N-gram Matching Analysis for Memorization

To examine memorization effects induced by different circuit representations, we perform an N-gram matching analysis on generated sequences. This analysis quantifies the extent to which the generator reproduces contiguous token patterns that appear verbatim in the training set.

For each experimental setting, we generate circuit sequences and extract the first 10 tokens and the last 10 tokens from each sequence, which are particularly sensitive to memorization under autoregressive decoding. We then scan the entire training set and compute the fraction of generated samples whose extracted token subsequences exactly match those of at least one training example.

When Circuit Type Tokens are not used, we generate 1,000 circuit sequences per model and perform N-gram matching over the full set of generated samples. This setting isolates the effect of circuit representation alone, independent of functional conditioning.

When Circuit Type Tokens are used, generation is performed in a type-conditioned manner. Specifically, for each circuit type, we generate approximately an equal number of samples, resulting in either 66 or 67 sequences per type such that the total number of generated samples is 1,000 across all 15 circuit types. N-gram matching is then conducted over the aggregated set of type-conditioned samples.

The analysis is conducted for both the device–pin-level graph representation and the proposed bipartite graph representation, enabling a direct comparison of memorization behavior under different representation schemes and conditioning settings.

G. Sequence-to-SPICE Translation Algorithm

Algorithm 2 translates a generated bipartite circuit sequence into an executable SPICE netlist by traversing the decoded graph structure. Specifically, the algorithm reconstructs device instances and net connections, assigns device terminals according to edge semantics, and infers supply, ground, and bias conditions from the connectivity patterns of the generated graph.

Algorithm 2 Sequence-to-SPICE Translation

Input: Generated bipartite sequence s.
Output: SPICE netlist or FAIL.
(1) ERC validation.
if $\mathrm{ERC}(s)$ fails **then**
  return FAIL.
end if
(2) Topology parsing.
Parse s into devices, external pins, and internal nets.
Construct topology graph G with typed edges; mark conducting edges(e.g., D/S/C/E).
(3) Terminal assignment.
for each typed edge incident to a device **do**
  Assign device terminals according to edge semantics (e.g., D/G/S/B, C/B/E, P/N).
end for
(4) Supply inference.
$V_{DD} \leftarrow$ shortest-path length between VDD and VSS over conducting edges.
(5) Bias assignment.
for each external port $p \notin \{\mathtt{VDD}, \mathtt{VSS}\}$ **do**
  Set bias using graph distance to VSS or VDD, selected by adjacent device polarity.
end for
(6) Netlist generation.
Emit SPICE code with device models, sources, DC operating point analysis, and output loads.
(7) Width refinement.
for each iteration until widths converge **do**
  for each MOSFET device **do**
    Identify same-type MOSFETs whose source nets are connected to the drain net of the device.
    Update the device width as the sum of widths of the identified MOSFETs.
    Clamp the updated width to a maximum value $W_{\max} = 500\,\mu\mathrm{m}$.
  end for
end for

H. Detailed Ablation Study Results by Circuit Type

Table 5. Detailed ablation study results by Circuit Type. For settings without the Circuit Type Token, all metrics except type classification accuracy are computed over 1,000 generated samples, including Validity, Novelty, Valid & Novel, and 10-gram matching.

	OpAmp	Mirror	Comp	Mix	LDO	Oscil	Filt	BGR	PAmp	VoltR	PConv	PLL	S Cap	A/D_D/A	General	Avg
Device-pin Representation + Type Token																
– 1000 samples per circuit type																
Validity (%)	81.7	89.6	62.8	74.6	50.3	74.2	66.4	86.6	78.3	76.1	90.4	79.9	70.7	65.9	79.8	**75.2**
Novelty (%)	42.1	34.9	58.3	46.8	72.6	45.4	47.6	35.2	42.5	39.9	23.9	41.7	52.6	49.3	45.3	**45.2**
Valid & Novel (%)	23.8	24.5	21.1	21.4	22.9	19.6	14.0	21.9	20.8	16.1	14.4	21.6	23.3	15.3	25.1	**20.4**
GAT Acc. (%)	99.4	95.7	95.9	99.5	99.2	98.5	96.8	99.2	94.8	95.5	99.7	99.1	99.2	98.6	–	**97.9**
10 Gram Matching (%)																**73.6**
Device-pin + Type Token + Renaming																
	OpAmp	Mirror	Comp	Mix	LDO	Oscil	Filt	BGR	PAmp	VoltR	PConv	PLL	S Cap	A/D_D/A	General	Avg
Validity (%)	63.0	78.7	26.2	60.4	21.5	46.4	32.2	44.7	53.6	35.0	29.4	53.4	39.4	31.1	60.5	**45.0**
Novelty (%)	82.2	77.2	94.8	92.4	95.7	89.5	96.5	82.6	95.1	92.3	91.5	90.4	88.0	94.6	87.6	**90.0**
Valid & Novel (%)	45.3	55.9	21.0	52.8	17.2	35.9	28.7	27.3	48.7	27.3	20.9	43.8	27.4	25.7	48.1	**35.1**
Classifier Result	96.5	83.4	91.6	99.4	89.5	94.7	92.6	88.9	83.5	70.5	89.1	89.6	95.3	94.3	–	**89.9**
10 Gram Matching (%)																**1.7**
Bipartite(Grammar) + Type Token																
	OpAmp	Mirror	Comp	Mix	LDO	Oscil	Filt	BGR	PAmp	VoltR	PConv	PLL	S Cap	A/D_D/A	General	Avg
Validity (%)	98.9	99.5	97.4	99.8	97.3	95.8	99.7	99.1	93.5	99.4	96.7	97.2	97.8	99.4	96.0	**97.8**
Novelty (%)	78.8	69.3	71.6	72.9	92.3	83.4	38.6	88.0	87.9	69.6	98.5	77.8	68.4	44.7	77.3	**74.6**
Valid & Novel (%)	77.7	68.8	69.0	72.7	89.6	79.2	38.3	87.1	81.4	69.0	95.2	75.0	66.2	44.1	73.3	**72.4**
GAT acc. (%)	97.7	79.1	88.8	99.9	99.9	97.7	96.3	99.9	79.1	85.1	100.0	88.2	81.3	97.1	–	**92.5**
10 Gram Matching (%)																**16.6**
Bipartite(Grammar) + Type Token + Renaming (Proposed)																
	OpAmp	Mirror	Comp	Mix	LDO	Oscil	Filt	BGR	PAmp	VoltR	PConv	PLL	S Cap	A/D_D/A	General	Avg
Validity (%)	97.6	99.4	97.1	99.3	95.6	97.7	98.9	97.7	97.7	98.5	96.2	97.8	95.9	99.6	98.1	**97.8**
Novelty (%)	89.1	77.4	92.0	84.1	97.6	92.1	92.9	94.0	99.7	91.0	99.2	90.9	91.4	96.5	92.9	**92.1**
Valid & Novel (%)	86.7	76.8	89.1	83.5	93.2	89.8	91.8	91.7	97.4	89.5	95.4	88.7	87.3	96.1	91.0	**89.9**
GAT acc. (%)	99.3	89.0	70.8	99.8	100.0	96.4	92.0	99.9	91.3	85.1	100.0	92.9	68.8	93.1	–	**91.3**
10 Gram Matching (%)																**0**

Table 5 reports detailed ablation results broken down by circuit type, complementing the aggregate analysis presented in Section 4.3. Across nearly all circuit categories, the bipartite representation combined with grammar-guided decoding consistently achieves high validity while maintaining substantially lower 10-gram matching rates than the device–pin representation, indicating reduced memorization at the per-type level.

Notably, this trend holds even for structurally diverse circuits such as oscillators, bandgap references, and phase-locked loops, suggesting that the proposed representation generalizes structural patterns across heterogeneous analog domains. In contrast, the device–pin representation exhibits pronounced trade-offs between validity and novelty when circuit-type tokens and renaming augmentation are applied, with performance varying significantly across circuit types. These per-type results further support the conclusion that the proposed bipartite formulation enables robust, controllable, and non-memorizing topology generation across a wide range of analog circuit classes.

I. Dataset Statistics

Table 6. Dataset Statistics

	OpAmp	LDO	BGR	PConv	Oscil	General	Mirror	Mix	PAmp	PLL	Filt	Comp	VoltR	S Cap	A/D_D/A	Total
Raw netlist	758	450	285	270	122	125	52	29	21	20	11	9	7	4	2	2165
(%)	35.0	20.8	13.2	12.5	5.6	5.8	2.4	1.3	1.0	0.9	0.5	0.4	0.3	0.2	0.1	100
Augmented sequence	132621	90000	55640	54000	22096	17322	7759	5154	3218	3787	1913	1800	1005	800	400	397515
(%)	33.4	22.6	14.0	13.6	5.6	4.4	2.0	1.3	0.8	1.0	0.5	0.4	0.3	0.2	0.1	100

Table 6 summarizes the composition of the dataset used in our experiments, as introduced in Section 4.1. The dataset consists of 2,165 raw SPICE netlists spanning 14 circuit types and an additional general category.

On this raw dataset, we apply sequence-level data augmentation strategies described in Appendix D, including the device renaming augmentation detailed in Section 3.4. These augmentations substantially increase the effective training set while preserving electrical equivalence, resulting in a total of 397,515 augmented sequences used for model training.