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

**Seungmin Kim**[1] **Mingun Kim**[2] **Yuna Lee**[2] **Yulhwa Kim**[3*]

[1]Department of Semiconductor and Display Engineering
[2]Department of Electrical and Computer Engineering
[3]Department of Semiconductor Systems Engineering
Sungkyunkwan University, Republic of Korea

{seungmin.kim, mingun.kim, yuna6548, yulhwakim}@skku.edu

## Abstract

Analog circuit design remains highly dependent on expert knowledge due to the complexity of device-level interactions and topology design. Recent transformer-based approaches for device-level topology generation have shown promise, yet they suffer from low electrical validity without human-in-the-loop (HITL) training and severe memorization caused by sequence-based circuit representations. In this work, we propose **AnalogToBi**, a framework for device-level analog circuit topology generation. AnalogToBi introduces *circuit-type conditioning* for categorizing heterogeneous multi-type topology datasets, *device renaming augmentation* to mitigate memorization, *a bipartite graph representation* for improved structural generalization, and *grammar-guided decoding* to enforce structural validity during bipartite graph generation. Experimental results demonstrate that AnalogToBi achieves high validity and novelty without HITL training while effectively avoiding memorization of training topologies. Our code and weights are available at https://github.com/Seungmin0825/AnalogToBi

## 1 Introduction

Analog circuits form the backbone of modern electronic systems, yet their design remains highly dependent on expert knowledge [1]. Analog circuit design typically involves determining a circuit topology that defines the core functionality of the circuit, followed by device sizing to achieve desired operating conditions [2]. This process requires deep understanding of device-level behavior and complex inter-device interactions, resulting in strong reliance on experienced analog designers. However, the limited availability of such expertise across the semiconductor industry has motivated growing interest in analog design automation.

Early analog design automation research primarily focused on device sizing optimization [3–10]. While effective for parameter refinement, these approaches assume that the circuit topology is given. Since circuit topology fundamentally determines analog circuit functionality, this assumption limits their applicability. Therefore, topology generation is a critical step toward fully automated analog circuit design.

Automating circuit topology design is substantially more challenging than device sizing optimization, as it requires modeling complex structural and electrical relationships among devices. As a result,

*Corresponding author

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

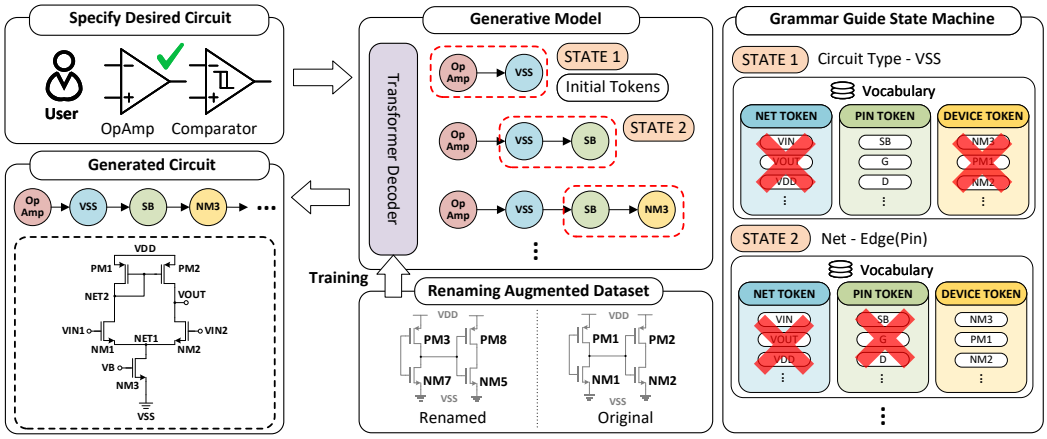



Figure 1: Overview of the proposed AnalogToBi framework. Given a circuit type, a Transformer decoder trained with renaming augmentation generates a device-level circuit sequence under grammar-guided decoding, which is converted into a bipartite device–net graph and netlist.

early analog design automation research largely avoided explicit topology generation despite its importance. Recent advances in transformer-based models, including large language models (LLMs), have renewed interest in this problem by enabling improved modeling of contextual information [11–13]. However, existing approaches still suffer from important limitations, including low electrical validity, memorization of training topologies, and reliance on HITL optimization. To address these challenges, we propose AnalogToBi, a novel framework for device-level analog circuit topology generation (Fig. 1).

## 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 and Topology Refinement

**Topology Search.** Previous works in this category aim to identify topologies that satisfy user-specified operational goals from predefined candidate sets manually designed based on prior analog circuit knowledge [14–17]. For example, FALCON [14] selects the most suitable topology for a given target specification. However, because these methods operate only within predefined candidate sets, they cannot generate topologies beyond those explicitly included.

**Topology Refinement.** Recent studies have explored the use of pre-trained LLMs for topology refinement, motivated by their strong capabilities in reasoning, code generation, and tool-integrated agentic workflows [18, 19]. Representative works include AnalogCoder [11, 20], Artisan [21], AnalogExpert [22], and AnalogSeeker [23]. In these approaches, users provide an initial circuit topology, typically represented in a code-like format such as a SPICE netlist, allowing pre-trained LLMs to interpret circuit structures. To further improve domain understanding, AnalogSeeker [23] additionally fine-tunes pre-trained LLMs for analog design tasks. The LLMs then act as agents that iteratively refine the topology using circuit simulation feedback. While effective for design refinement, these approaches remain highly dependent on the quality of the initial topology, limiting their ability to systematically explore novel circuit structures.

### 2.2 Behavioral-Level Topology Generation

Early topology generation approaches primarily adopted behavior-level topology generation strategies [24–26]. In these approaches, key analog circuit components are abstracted as behavior-level

building blocks whose types and internal designs are predefined. Topology generation is then performed by composing and connecting these abstract blocks. For example, CktGNN [26] 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.

### 2.3 Device-Level Topology Generation

More recently, active research has explored device-level topology generation to enable fully flexible analog circuit design [12, 27, 13, 28, 29]. 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 rather than relying on pre-trained language models. LaMAGIC [12, 27], a pioneering work in this category, proposes training a new model for device-level topology generation. While effective within its target domain, it is limited to a single circuit type, specifically power converters.

AnalogGenie [13, 28] extends this line of work by training a single model on a dataset covering multiple circuit types, enabling broader topology generation. However, to train a high-quality topology generation model that produces a high ratio of electrically valid circuits, AnalogGenie relies on HITL 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. Moreover, its device-pin-level graph representation encourages memorization rather than structural generalization.

In summary, enabling fully flexible analog circuit topology design ultimately requires device-level topology generation. To overcome the limitations of existing approaches, there is a clear need for a topology generation framework that ensures high electrical validity without HITL training and promotes topological novelty without memorization.

## 3 Proposed Method

We propose AnalogToBi, a novel device-level analog circuit topology generation framework. Similar to recent approaches [13, 28], 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 Conditioning and Data Preprocessing

As discussed in Section 2.3, AnalogGenie [13, 28] is trained on a dataset covering multiple circuit types to support broader topology generation. However, circuits from different functional categories exhibit fundamentally different topological structures. From the perspective of generative modeling, such an unstructured dataset induces a multimodal distribution over circuit topologies. For example, as shown in Fig. 2, OpAmps typically employ a differential pair with a current mirror load, optionally augmented with cascode stages or extended to two-stage configurations. In contrast, comparators often rely on latch-based structures or pre-amplifier plus high gain stage architectures. These examples highlight that

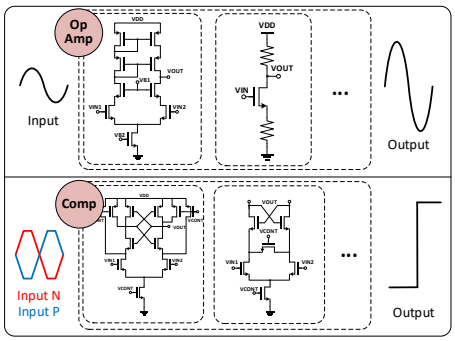



Figure 2: Example analog circuit topologies for two circuit types: OpAmp and comparator (Comp).

circuit topology is tightly coupled with functionality, and that different circuit types follow distinct structural rules.

Training a generative model on such heterogeneous data without explicit conditioning can lead to undesirable behavior. In particular, the model may interpolate between distinct modes, resulting in physically invalid circuits that violate underlying design principles. To address this, we use a preprocessed training dataset in which circuits are categorized into functional types, thereby providing conditional constraints that guide the model toward electrically valid topologies.

Following widely adopted functional categorizations in analog design, such as OpAmps and comparators, the 2,165 raw netlists from the AnalogGenie dataset are categorized into 15 circuit types (see Appendix A for details). We further incorporate a circuit-type token into the model vocabulary, which is provided as the first input token to explicitly condition the generation process.

However, naively introducing data preprocessing and circuit-type tokens into the AnalogGenie framework exacerbates the memorization problem. This primarily stems from data sparsity, as partitioning the dataset into multiple circuit types reduces the number of training samples available for each category. With fewer examples per type, the model tends to overfit to typical patterns rather than generalizing to diverse topologies. To mitigate this issue, we introduce a renaming data augmentation strategy in the following section.

### 3.2 Data Augmentation with Device Renaming

Transformer-based circuit generation models operate on sequential representations, requiring each circuit to be converted into a token sequence. Each circuit is first transformed into a graph representation and then serialized into a sequence based on a graph traversal order. Because different traversal orders produce different sequences for the same circuit graph, previous work [13, 28] introduces a data augmentation strategy that generates multiple sequence variants by applying different traversal orderings. While this traversal-order augmentation increases the number of training sequences, it does not prevent the model from memorizing sequence patterns, as the original device indices from the raw dataset are repeatedly exposed across augmented variants. To address this limitation, we introduce an additional *device renaming augmentation* that randomly reassigns device identifiers across training samples. This augmentation exploits the fact that device identifiers encode instance names rather than functional behavior: renaming a device token such as `NMOS1` to `NMOS7` does not alter the electrical functionality or connectivity of the circuit (see appendix C for details). By randomizing device indices, this augmentation disrupts index-based pattern memorization while preserving electrical equivalence, thereby encouraging the model to learn structurally grounded representations rather than surface-level token patterns.

Our evaluation shows that device renaming augmentation effectively alleviates memorization, even under circuit-type conditioning. However, when applied to the original AnalogGenie framework, the improvement in novelty comes at the cost of degraded validity (a detailed quantitative analysis is provided in Section 4.3). This trade-off stems from the underlying circuit representation used

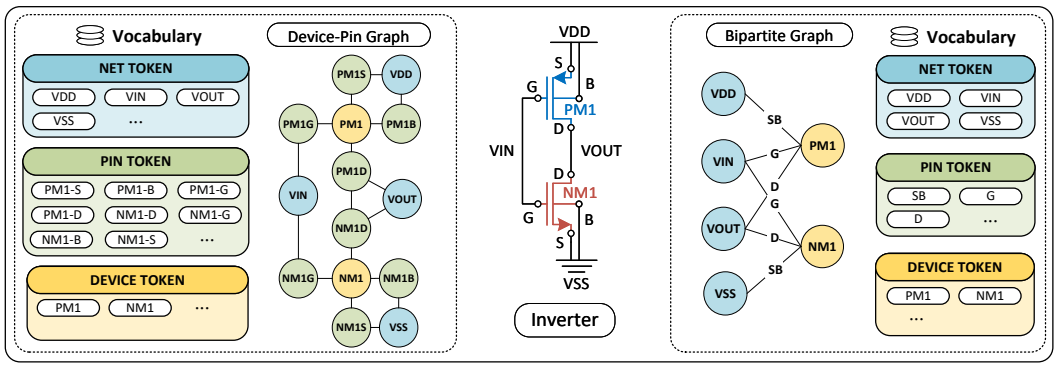



Figure 3: Comparison between conventional device–pin level representation [13] and the proposed bipartite graph representation. The device–pin representation (left) treats each device pin as a separate token, while the bipartite representation (right) models devices and nets as distinct node types with pin semantics encoded as typed edges.

in AnalogGenie. 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. It supports circuit generation with up to 35 NMOS and 35 PMOS devices (e.g., NM1, ..., NM35 and PM1, ..., PM35), which are further expanded into pin-level tokens such as NM1S, NM1G, NM1D, and NM1B, where S, G, D, and B denote source, gate, drain, and body, respectively. This results in a large and fixed token space with $70 \times 4$ distinct pin-specific tokens. While this representation explicitly encodes pin-level connectivity in a sequential format, it effectively reduces circuit generation to an ordering problem over a fixed token set. Instead of learning the underlying connectivity structure, the model is encouraged to learn how to arrange a predefined set of tokens into valid sequences. As a result, the model tends to memorize token-level patterns rather than learning generalizable structural relationships. Consequently, validity in the original framework is largely maintained through memorization, and disrupting this memorization via augmentation leads to a degradation in validity. This reveals a fundamental limitation of the representation, where validity and novelty are inherently entangled through memorization.

### 3.3 Bipartite Graph Circuit Representation

To encourage the model to learn generalizable topology patterns rather than memorizing training sequences, we propose a more flexible representation that decouples device-specific identifiers from functional connectivity. To this end, we leverage the inherent structural components 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 detailed in Appendix D. In this representation, 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.

### 3.4 Grammar-Guided Decoding for Bipartite Graph Representation

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 [30, 31], 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 $T-1$.

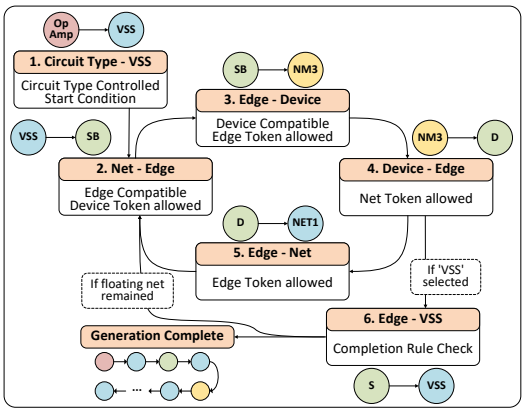



Figure 4: Grammar-guided decoding state machine.

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 F.

# 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 [13]. To ensure stable model training, we use the functionally categorized dataset described in Section 3.1. For each circuit category, 90% of the samples 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 (Section 3.2), 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 device-level topology generation, including LaMAGIC [12], and AnalogGenie(-Lite) [13, 28]. These methods differ from AnalogToBi in terms of circuit representation, generation capability, and scalability, as discussed in Section 2. LaMAGIC focuses on generating power converter circuits within a single circuit domain, resulting in highly limited topology exploration. AnalogGenie serves as the strongest baseline for our task, as it supports the generation of multiple circuit types. However, it requires HITL feedback to achieve high electrical validity and suffers from severe memorization of training circuits. 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 3.3. 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 $3 \times 10^{-4}$ and a batch size of 64 for 100,000 iterations. The entire training process takes 22.8 hours on an NVIDIA RTX 5880

Table 1: Comparison of AnalogToBi with baselines. *Model(#param)* denotes the model architecture and number of parameters. *HITL* indicates HITL involvement during training, and *#Type* denotes the number of circuit types in the training dataset. All baseline results are taken from the original papers except for AnalogGenie*, which is reproduced using the officially released implementation. Novelty marked with $^+$ corresponds to the value reported in the original AnalogGenie paper, whose evaluation methodology was not explicitly described.

| | **Properties** | | | **Evaluation Results** | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **Method** | Model (#param.) | HITL | #Type | Validity↑ | Novelty↑ | Val.&Nov.↑ | Exact.↓ | Div.↑ | GED↑ |
| LaMAGIC [12] | Flan-T5 (250M) | × | 1 | 96.0% | – | – | – | – | |
| AnalogGenie [13] | Custom (11.8M) | × | 11 | 73.5% | 98.9%$^+$ | – | – | – | |
| AnalogGenie [13] | Custom (11.8M) | ○ | 11 | 93.2% | 99.0%$^+$ | – | – | – | |
| AnalogGenie-Lite [28] | Custom (11.8M) | ○ | 11 | 97.0% | 99.8%$^+$ | – | – | – | |
| AnalogGenie* [13] | Custom (11.8M) | × | 11 | 75.6% | 50.4% | 26.0% | 39.6% | 26.0% | 25.2 |
| **AnalogToBi (Proposed)** | **Custom (11.3M)** | × | **15** | **97.8%** | **92.1%** | **89.9%** | **0%** | **75.3%** | **93.6** |

GPU. During inference, circuits are generated autoregressively using multinomial sampling with a temperature of $\tau = 0.7$.

**Metrics.** To properly evaluate the ability of the circuit generation model to generate structurally correct and novel circuit topologies, we assess performance using six metrics: Validity, Novelty, Valid & Novel, Exact sequence matching (Exact.), Diversity (Div.), Graph Edit Distance (GED). **Validity** measures the fraction of generated circuits that are electrically valid, such that they can be converted into SPICE netlists without errors. While this metric alone does not guarantee that the generated circuits function properly, previous works on device-level topology generation has primarily adopted this level of validity evaluation to assess topology quality [13, 28]. For fair comparison, we follow the same evaluation protocol. We note that improving the ratio of fully correctly operating generated circuits remains an important future direction for device-level topology generation research. **Novelty** measures the fraction of generated circuits whose topologies are not observed in the training set. We evaluate novelty using graph isomorphism, where a generated circuit is considered novel if its topology is not isomorphic to any training circuit. Although prior works [13, 28] also reported novelty metrics, their exact evaluation methodology was not explicitly described. Therefore, the novelty values reported in our work may not be directly comparable to those reported in previous studies. **Valid & Novel** measures the proportion of generated circuits that are both electrically valid and novel. **Exact.** indicates the rate at which the transformer decoder model outputs the same sequence as the dataset used when learning. A high Exact. score indicates that the model reproduces training samples verbatim rather than generating unseen structures. Therefore, both Novelty and Exact. serve as indicators of the memorization tendency of the model. **Div.** measures output diversity by reporting $d/n$, where $d$ is the number of distinct valid-and-novel topologies among $n$ generated samples. **GED** denotes the average minimum graph edit distance between each generated topology and the topologies in the training dataset [32–35]. It measures the structural difference between generated circuits and training samples. A higher GED indicates that the model generates structurally more diverse circuits rather than producing minor variations of existing training topologies. GED values can be affected by how graph nodes are matched. Matching nodes based on device indices and net names may yield nonzero distances even between isomorphic graphs. To avoid this issue, we employ canonical graph representations that ensure a GED of zero for isomorphic graphs.

### 4.2 Main Results

Table 1 presents a comprehensive comparison between AnalogToBi and prior approaches in terms of both methodological properties and generation quality. LaMAGIC achieves high validity (96.0%) without HITL training or inference. However, it is limited to single-type topology generation, specifically power converters, which restricts its applicability to general analog circuit design. AnalogGenie(-Lite) extends topology generation to multiple circuit types. However, without HITL-based expert feedback during training, its validity drops to 73.5%. Achieving validity above 90% requires HITL training, where expert designers evaluate generated circuits and provide feedback throughout optimization. For fair comparison, we compare AnalogToBi against AnalogGenie without HITL, representing a fully automated training setting. Because the novelty evaluation methodology of AnalogGenie was not explicitly described, we reproduced the original model using the officially

Table 2: Ablation study of AnalogToBi evaluating the impact of four key design components. The results highlight how device renaming augmentation and grammar-guided decoding, together with the bipartite representation, overcome the validity–novelty trade-off observed in the conventional approach.

| **Options** | | | | **Evaluation Results** | | | |
|---|---|---|---|---|---|---|---|
| Representation | Type Token | Rename | Grammar | Validity ↑ | Novelty ↑ | Val.&Nov. ↑ | Exact. ↓ |
| Device-pin | × | × | – | 75.6% | 50.4% | 26.0% | 39.6% |
| Device-pin | ○ | × | – | 86.1% | 29.2% | 15.4% | 57.5% |
| Device-pin | ○ | ○ | – | 55.6% | 84.1% | 39.7% | 0% |
| Bipartite | × | × | × | 53.6% | 61.0% | 15.3% | 0% |
| Bipartite | × | × | ○ | 61.8% | 57.3% | 19.9% | 0% |
| Bipartite | ○ | × | ○ | 97.8% | 74.6% | 72.4% | 1.4% |
| **Bipartite** | ○ | ○ | ○ | **97.8%** | **92.1%** | **89.9%** | **0%** |

released implementation and evaluated it using our metrics. The reproduced model achieves a validity of 75.6%, which is consistent with the originally reported result (73.5%). We denote this reproduced model as AnalogGenie*.

AnalogToBi consistently outperforms AnalogGenie* across all evaluation metrics. Specifically, AnalogToBi achieves a validity of 97.8%, a novelty of 92.1%, and an exact-match rate of 0%, indicating that the model does not memorize training sequences. Furthermore, the high Valid & Novel ratio of 89.9% demonstrates that the generated circuits remain highly novel even when considering only electrically valid samples. This is important because novelty alone can be artificially increased by generating random invalid sequences. The results therefore indicate that AnalogToBi successfully learns generalizable topology patterns and generates novel yet electrically valid circuits. Moreover, AnalogToBi achieves a Diversity of 75.3% and a GED of 93.6, compared with 26.0% and 25.2 for AnalogGenie*, respectively. Although some valid and novel topologies generated by AnalogToBi are repeated, it still produces nearly three times as many distinct valid and novel topologies per generated sample as AnalogGenie*, demonstrating greater diversity in valid topology generation.

### 4.3 Ablation Study

Table 2 presents an ablation study of the proposed AnalogToBi framework, evaluating the impact of four key design components: circuit-type tokens, device renaming augmentation, circuit representation, and grammar-guided decoding. Please note that the device–pin representation without circuit-type tokens or renaming augmentation corresponds to the original AnalogGenie framework. When circuit-type tokens are introduced into the device–pin representation, validity improves from 75.6% to 86.1%. However, novelty drops from 50.4% to 29.2%, and Exact. increases from 39.6% to 57.5%. This indicates that the device–pin representation tends to achieve higher validity by memorizing training samples rather than learning generalizable topology patterns. Applying device renaming augmentation alleviates this memorization behavior, increasing novelty to 84.1% while reducing Exact. to 0%. However, this improvement comes at the cost of validity, which drops significantly to 55.6%. These results suggest that the device–pin representation fundamentally lacks the ability to learn topology patterns without relying on memorization.

For the proposed bipartite graph representation, validity remains relatively low without grammar-guided decoding because learning the structural rules required for bipartite graph is challenging under limited training data. By introducing grammar-guided decoding, validity improves by 8.2%p, demonstrating that explicit structural constraints effectively assist the generation process. Interestingly, without circuit-type tokens, the bipartite representation achieves lower validity than the device–pin representation while exhibiting substantially lower Exact. scores. This indicates that the bipartite representation is less prone to memorization and instead attempts to learn topology-level structural patterns. However, because the training dataset contains structurally heterogeneous circuit topologies without functional categorization, the bipartite representation struggles to achieve high validity. Finally, when the bipartite representation is combined with circuit-type conditioning, validity increases substantially to 97.8%. Moreover, applying renaming augmentation on top of this configuration preserves validity while further increasing novelty to 92.1%.

### 4.4 Analysis of Device Index Memorization

To investigate whether device renaming augmentation during training mitigates index-based memorization, we evaluate the trained models on unseen circuit sequences before and after randomly permuting their device indices. Since device indices serve only as instance identifiers, this permutation preserves the underlying circuit topology.

Table 3: Perplexity and mean NLL per token class on unseen sequences, with original or randomly permuted device indices.

| | | | NLL per token | | |
|---|---|---|---|---|---|
| Model Training | Indices | PPL | device | edge | net |
| w/o renaming | original | 1.32 | 0.34 | 0.34 | 0.09 |
| | permuted | 576.52 | 13.81 | 2.58 | 6.45 |
| w/ renaming | original | 2.05 | 1.74 | 0.44 | 0.26 |
| | permuted | 2.04 | 1.73 | 0.44 | 0.26 |

Table 3 reports the perplexity (PPL) and token-wise negative log-likelihood (NLL) before and after device index permutation. Without renaming augmentation, PPL increases dramatically from 1.32 to 576.52, exceeding the perplexity of uniform random prediction over the 397-token vocabulary. Moreover, NLL increases not only for device tokens but also for edge and net tokens, indicating that the model heavily relies on index-specific sequence patterns rather than underlying circuit connectivity. In contrast, the model trained with renaming augmentation exhibits nearly identical PPL before and after permutation (2.05 vs. 2.04), with consistently stable NLL across all token categories. These results demonstrate that device renaming augmentation effectively eliminates sensitivity to arbitrary device indices, encouraging the model to learn connectivity-dependent patterns rather than memorizing index-specific token sequences.

Table 4: Breakdown of AnalogToBi generation results by circuit type. (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. (%) | 95.8 | 92.4 | 86.4 | 99.6 | 100.0 | 96.7 | 93.6 | 100.0 | 95.3 | 98.5 | 100.0 | 89.6 | 95.2 | 95.1 | – | **95.6** |

### 4.5 Additional Analysis on Circuit-Type Controllability

As the training dataset is categorized by circuit type and circuit-type tokens are introduced during training, AnalogToBi has the potential to control the type of generated topologies through circuit-type conditioning. To evaluate whether generated circuits match the intended category, we train a Graph Attention Network (GAT) classifier that predicts the circuit type of generated topologies. We define type accuracy as the fraction of generated circuits whose predicted type matches the input circuit-type token. Details of the GAT architecture and training procedure are provided in Appendix E.2.

Table 4 reports the circuit-type-wise breakdown of generated circuits, where 1,000 samples are generated for each category. The overall average type accuracy reaches 95.6%. These results demonstrate that AnalogToBi can generate topologies consistent with the intended circuit category.

### 4.6 Case Study with SPICE Simulation

While the current goal of device-level topology generation is to generate electrically valid circuits without evaluating their operational behavior, we further conduct SPICE-based case studies on generated topologies. Circuit-type conditioning facilitates systematic SPICE evaluation, as each generated topology has a predefined circuit type (Section 3.1), allowing the corresponding testbench to be automatically assigned. In contrast, prior methods [13, 28] generate circuits unconditionally, requiring an additional step to identify the function of each generated circuit before selecting an appropriate testbench. For this evaluation, we translate the valid and novel topologies among the 1,000 samples generated for each circuit type into SPICE netlists by recovering device terminals from the pin-labeled edges of the bipartite sequences. We then simulate all netlists whose ports match the corresponding testbenches using Cadence Spectre with the GPDK 45 nm CMOS process [36].

Specifically, we perform AC and transient analyses for OpAmps, transient analyses for comparators and oscillators, and DC sweeps of load current and input voltage for LDOs. The netlist translation code and testbenches are included in our code release.

Fig. 5 presents the best-performing OpAmp among the simulated candidates, ranked by FoM. The circuit is evaluated under a 3.3V supply and a capacitive load of 100 pF using transient and AC analyses. The generated OpAmp achieves a FoM (MHz · pF/mW) of 728.1, substantially outperforming the reported result of AnalogGenie (FoM 36.5). Even under default sizing and biasing conditions, the circuit achieves a high DC gain of 56.8 dB and moderate bandwidth characteristics (BW 58.7 kHz and GBW 40.6 MHz), exceeding the best reported GBW of AnalogGenie (12 MHz). These results suggest that, by generating diverse and electrically valid circuits (Table 1), AnalogToBi can potentially discover practically competitive analog topologies. Additional case studies are provided in Appendix H.

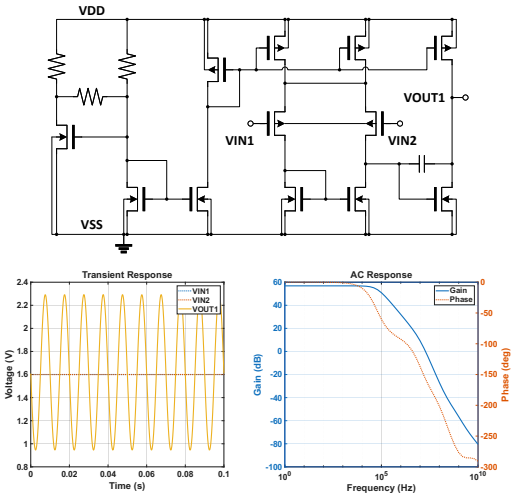



Figure 5: Generated OpAmp topology and transient, AC simulation result. For the transient simulation, 1 $mV_{pp}$ sinusoidal input was applied.

## 5 Limitation and Future Work

While existing device-level topology generation research primarily focuses on generating electrically valid circuits, extending these models toward properly operating circuit generation remains an important future direction. A key limitation of prior approaches is the lack of explicit circuit-type control, which makes it difficult to define circuit-specific simulation conditions and performance objectives within a unified framework. As a result, previous studies have largely focused on connectivity-level validity rather than operational behavior. In contrast, AnalogToBi introduces circuit-type conditioning through circuit-type tokens, enabling generated topologies to be associated with well-defined functional categories. This provides a natural foundation for integrating circuit-type-specific simulation pipelines into training. For example, different simulation setups and objectives can be automatically assigned depending on the target circuit type, such as gain and bandwidth for OpAmps or switching characteristics for comparators. Building on this capability, future work could incorporate simulation feedback directly into the optimization loop, enabling topology generation models to optimize not only electrical validity but also operational correctness and circuit performance.

## 6 Conclusion

In this work, we propose AnalogToBi, a novel framework for device-level analog circuit topology generation. By integrating circuit-type conditioning, device renaming augmentation, a bipartite graph representation, and grammar-guided decoding, AnalogToBi addresses the memorization issue and low electrical validity that limit existing approaches. Experimental results demonstrate that AnalogToBi enables fully automated training without any HITL intervention while achieving both high validity and high novelty in generated circuit topologies.

## Acknowledgments

This work was supported in part by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (RS-2026-25507427: Development of Efficient Architectures and Training Techniques for High-Performance Lightweight AI Models, RS-2025-10692981: AI Semiconductor Innovation Research Center (Sungkyunkwan University)), K-CHIPS (Korea Collaborative & High-tech Initiative for Prospective Semiconductor Research) funded by the Ministry of Trade, Industry and Resources (MOTIR, Korea) (RS-2026-25521189: Development of Application Processor Design Technologies for Multimodal Sensor

Signal Processing), and AI Computing Infrastructure Enhancement (GPU Rental Support) User Support Program funded by MSIT (No.RQT-25-090058).

## References

[1] Konstantinos G Liakos and Fotis Plessas. Analog design and machine learning: A review. *Electronics*, 14 (17):3541, 2025.

[2] Chang Liu and Danial Chitnis. Eesizer: Llm-based ai agent for sizing of analog and mixed signal circuit. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 73(6):3825–3834, 2026. doi: 10.1109/TCSI.2025.3646359.

[3] Jinhai Hu, Wang Ling Goh, and Yuan Gao. Ai-powered agile analog circuit design and optimization. *arXiv preprint arXiv:2505.03750*, 2025.

[4] Weonhyeog Lee and Ickhyun Song. Machine-learning-based design automation for optimizing analog/rf circuit applications. In *2023 8th IEEE International Conference on Network Intelligence and Digital Content (IC-NIDC)*. IEEE, 2023.

[5] Weidong Cao, Mouhacine Benosman, Xuan Zhang, and Rui Ma. Domain knowledge-infused deep learning for automated analog/radio-frequency circuit parameter optimization. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022.

[6] Keertana Settaluri, Ameer Haj-Ali, Qijing Huang, Kourosh Hakhamaneshi, and Borivoje Nikolic. Autockt: Deep reinforcement learning of analog circuit designs. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2020.

[7] Hanrui Wang, Kuan Wang, Jiacheng Yang, Linxiao Shen, Nan Sun, Hae-Seung Lee, and Song Han. Gcn-rl circuit designer: Transferable transistor sizing with graph neural networks and reinforcement learning. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2020.

[8] Wenlong Lyu, Pan Xue, Fan Yang, Changhao Yan, Zhiliang Hong, Xuan Zeng, and Dian Zhou. An efficient bayesian optimization approach for automated optimization of analog circuits. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 65(6):1954–1967, 2017.

[9] Han Young Koh, Carlo H Sequin, and Paul R Gray. Opasyn: A compiler for cmos operational amplifiers. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 9(2):113–125, 1990.

[10] Emil S Ochotta, Rob A Rutenbar, and L Richard Carley. Synthesis of high-performance analog circuits in astrx/oblx. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(3): 273–294, 1996.

[11] Yao Lai, Sungyoung Lee, Guojin Chen, Souradip Poddar, Mengkang Hu, David Z Pan, and Ping Luo. Analogcoder: Analog circuit design via training-free code generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, 2025.

[12] Chen-Chia Chang, Yikang Shen, Shaoze Fan, Jing Li, Shun Zhang, Ningyuan Cao, Yiran Chen, and Xin Zhang. Lamagic: language-model-based topology generation for analog integrated circuits. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24, 2024.

[13] Jian Gao, Weidong Cao, Junyi Yang, and Xuan Zhang. Analoggenie: A generative engine for automatic discovery of analog circuit topologies. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=jCPak79Kev`.

[14] Asal Mehradfar, Xuzhe Zhao, Yilun Huang, Emir Ceyani, Yankai Yang, Hamidreza Aghasi, Salman Avestimehr, et al. Falcon: An ml framework for fully automated layout-constrained analog circuit design. *Advances in Neural Information Processing Systems*, 2026.

[15] Andreas Gerlach, Juergen Scheible, Thoralf Rosahl, and Frank-Thomas Eitrich. A generic topology selection method for analog circuits demonstrated on the ota example. In *2015 11th Conference on Ph. D. Research in Microelectronics and Electronics (PRIME)*. IEEE, 2015.

[16] Souradip Poddar, Ahmet Budak, Linran Zhao, Chen-Hao Hsu, Supriyo Maji, Keren Zhu, Yaoyao Jia, and David Z Pan. A data-driven analog circuit synthesizer with automatic topology selection and sizing. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2024.

[17] Petar Veselinovic, Domine Leenaerts, W Van Bokhoven, Francky Leyn, Frank Proesmans, Georges Gielen, and Willy Sansen. A flexible topology selection program as part of an analog synthesis system. In *Proceedings the European Design and Test Conference. ED&TC 1995*. IEEE, 1995.

[18] Chengjie Liu, Yijiang Liu, Yuan Du, and Li Du. LADAC: Large language model-driven auto-designer for analog circuits. TechRxiv preprint, 2024.

[19] Jitendra Bhandari, Vineet Bhat, Yuheng He, Hamed Rahmani, Siddharth Garg, and Ramesh Karri. Masala-chai: A large-scale spice netlist dataset for analog circuits by harnessing ai. *arXiv preprint arXiv:2411.14299*, 2024.

[20] Yao Lai, Souradip Poddar, Sungyoung Lee, Guojin Chen, Mengkang Hu, Bei Yu, Ping Luo, and David Z Pan. Analogcoder-pro: Unifying analog circuit generation and optimization via multi-modal llms. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2026.

[21] Zihao Chen, Jiangli Huang, Yiting Liu, Fan Yang, Li Shang, Dian Zhou, and Xuan Zeng. Artisan: Automated operational amplifier design via domain-specific large language model. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024.

[22] Haoyi Zhang, Shizhao Sun, Yibo Lin, Runsheng Wang, and Jiang Bian. Analogxpert: Automating analog topology synthesis by incorporating circuit design expertise into large language models. In *2025 International Symposium of Electronics Design Automation (ISEDA)*. IEEE, 2025.

[23] Zihao Chen, Ji Zhuang, Jinyi Shen, Xiaoyue Ke, Xinyi Yang, Mingjie Zhou, Zhuoyao Du, Xu Yan, Zhouyang Wu, Zhenyu Xu, et al. AnalogSeeker: An open-source foundation language model for analog circuit design. *arXiv preprint arXiv:2508.10409*, 2025.

[24] Zhenxin Zhao and Lihong Zhang. Graph-grammar-based analog circuit topology synthesis. In *2019 IEEE International Symposium on Circuits and Systems (ISCAS)*. IEEE, 2019.

[25] Jialin Lu, Liangbo Lei, Fan Yang, Li Shang, and Xuan Zeng. Topology optimization of operational amplifier in continuous space via graph embedding. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2022.

[26] Zehao Dong, Weidong Cao, Muhan Zhang, Dacheng Tao, Yixin Chen, and Xuan Zhang. CktGNN: Circuit graph neural network for electronic design automation. In *International Conference on Learning Representations (ICLR)*, 2023.

[27] Chen-Chia Chang, Wan-Hsuan Lin, Yikang Shen, Yiran Chen, and Xin Zhang. Lamagic2: advanced circuit formulations for language model-based analog topology generation. In *Proceedings of the 42nd International Conference on Machine Learning*, ICML'25, 2025.

[28] Jian Gao, Weidong Cao, and Xuan Zhang. Analoggenie-lite: Enhancing scalability and precision in circuit topology discovery through lightweight graph modeling. In *Forty-second International Conference on Machine Learning*, 2025. URL `https://openreview.net/forum?id=KRkOWTIIOI`.

[29] Qiufeng Li, Shu Hong, Jian Gao, Xuan Zhang, Tian Lan, and Weidong Cao. Analogfed: Federated discovery of analog circuit topologies with generative ai. *arXiv e-prints*, 2025.

[30] Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. Grammar-constrained decoding for structured nlp tasks without finetuning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

[31] Federico Raspanti, Tanir Ozcelebi, and Mike Holenderski. 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)*, 2025.

[32] Horst Bunke and Kim Shearer. A graph distance metric based on the maximal common subgraph. *Pattern recognition letters*, 19(3-4):255–259, 1998.

[33] Panagiotis Papadimitriou, Ali Dasdan, and Hector Garcia-Molina. Web graph similarity for anomaly detection. *Journal of Internet Services and Applications*, 1(1):19–30, 2010.

[34] Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

[35] Andreas Fischer, Kaspar Riesen, and Horst Bunke. Improved quadratic time approximation of graph edit distance by combining hausdorff matching and greedy assignment. *Pattern Recognition Letters*, 87:55–62, 2017.

[36] Tomal Suvro Sannyashi, Sk Babul Akhtar, Tanmay Sinha Roy, Biswajit Jana, and Soumen Ghosh. Performance analysis of differential amplifiers using cadence with gpdk45, gpdk90, and gpdk180. In *2025 Devices for Integrated Circuit (DevIC)*, pages 68–73. IEEE, 2025.

[37] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[38] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[39] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations*, 2018.

# A Dataset Statistics

Table 5: Raw dataset and augmented 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 |

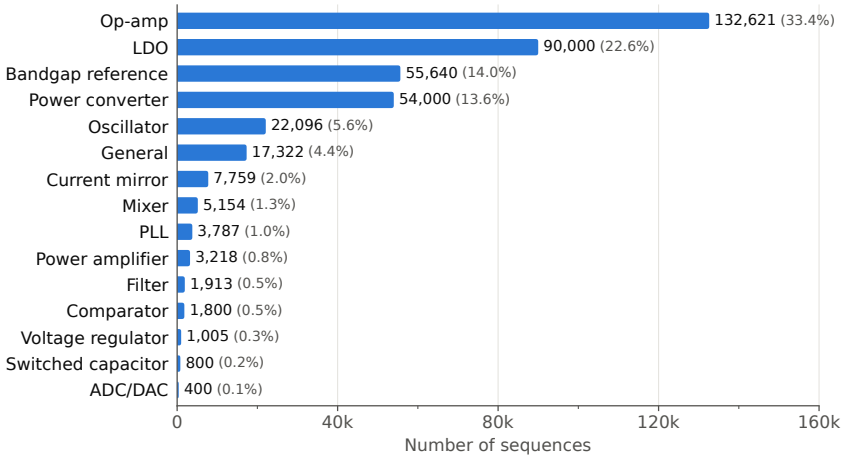



Figure 6: Augmented dataset statistics

Table 5 and Fig. 6 summarize 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 3.2. 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.

Table 6: 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,NM2, PM2, NPN1, R1, C1, L1, DIO1 |
| Net Token | VSS, VDD, NET1, NET2, VIN1, VOUT1, IIN1, IB1 |
| Edge (Pin) Token | M_G, M_S, M_SB, M_GD, B_C, B_E, R_C, C_C, L_C |
| Termination Token | TRUNCATE |

# B Vocabulary Definition

Table 6 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.

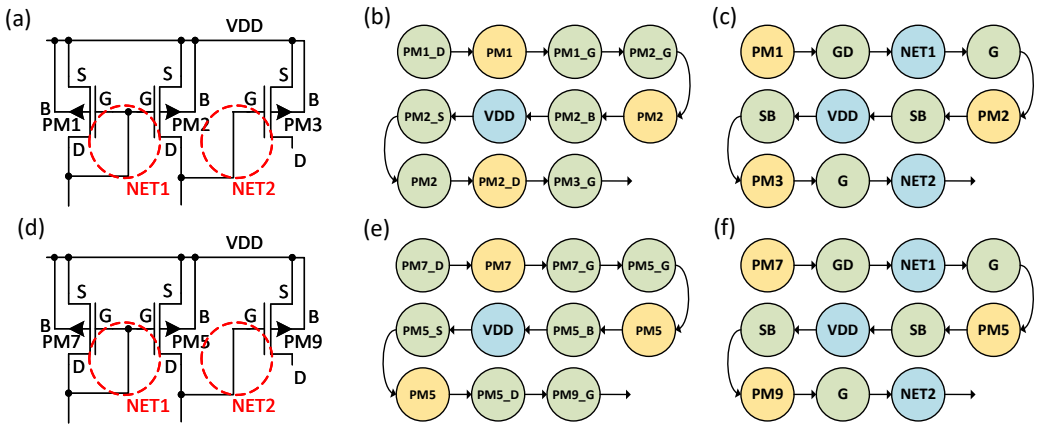



Figure 7: Examples of device renaming augmentation applied to circuit graphs serialized into token sequences. (a), (d) show the original and renamed circuit graphs. (b), (e) show the renamed sequences under the device–pin representation, and (c), (f) show the renamed sequences under the bipartite representation. In both representations, only the token index assigned to each device are randomly shuffled while the underlying structural connectivity remains unchanged.

## C Examples of Renaming Augmentation

Fig. 7 illustrates examples of the renaming augmentation applied to circuit graphs serialized into token sequences. Device renaming augmentation can be applied to both the device–pin representation (b), (e) and the bipartite representation (c), (f). In both cases, only the token indices assigned to each device are randomly shuffled, while the underlying structural connectivity of the circuit remains unchanged, as shown in the original circuits (a), (d). As such, renaming augmentation generates structurally equivalent but tokenically distinct sequences, encouraging the model to focus on learning circuit topology rather than memorizing specific token patterns.

**Algorithm 1** Structure-Preserving Data Augmentation for Bipartite Circuit Sequences

**Input:** Bipartite circuit graph $G = (V, E)$ with typed edges; reference node $v_{\mathrm{ref}} = \texttt{VSS}$; maximum number of sequences $K$; attempt budget $A$; maximum sequence length $T_{\max}$.
**Output:** Set $\mathcal{S}$ of at most $K$ distinct augmented sequences.
Initialize $\mathcal{S} \leftarrow \emptyset$.
**while** $|\mathcal{S}| < K$ **and** $a < A$ **do**
  $a \leftarrow a + 1$.
  Initialize $s \leftarrow [v_{\mathrm{ref}}]$, $v \leftarrow v_{\mathrm{ref}}$, $\mathcal{E}_{\mathrm{vis}} \leftarrow \emptyset$, $\mathcal{V}_{\mathrm{vis}} \leftarrow \{v_{\mathrm{ref}}\}$.
  **while** $\mathcal{E}_{\mathrm{vis}} \neq E$ **do**
    Scan the incident edges of $v$ in random order and choose the next move by priority:
      (i) an unvisited incident edge $e = (v, u)$;
      (ii) a shortest path from $v$ to a node incident to an unvisited edge (nodes and edges may be revisited).
    For every traversed edge $e = (v, u)$, append $\tau(e)$ and $u$ to $s$, update $\mathcal{E}_{\mathrm{vis}} \leftarrow \mathcal{E}_{\mathrm{vis}} \cup \{e\}$, $\mathcal{V}_{\mathrm{vis}} \leftarrow \mathcal{V}_{\mathrm{vis}} \cup \{u\}$, and set $v \leftarrow u$.
  **end while**
  **Close the loop:** if $v \neq v_{\mathrm{ref}}$, append the shortest path from $v$ back to $v_{\mathrm{ref}}$.
  **Coverage check:** if $\mathcal{V}_{\mathrm{vis}} \neq V$ or $s \in \mathcal{S}$, discard $s$ and **continue**.
  **Length check:** if $|s| \geq T_{\max}$, discard $s$ and **continue**.
  **ERC check:** if $\mathrm{ERC}(s)$ fails (e.g., floating internal nets or incomplete device connections), discard $s$ and **continue**.
  Pad $s$ with `TRUNCATE` to length $T_{\max}$ and add it to $\mathcal{S}$.
**end while**
**return** $\mathcal{S}$.

# 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.

Here, $G = (V, E)$ denotes a bipartite circuit graph with device and net nodes, $\tau(e)$ denotes the discrete edge-type token associated with edge $e$ and $s$ denotes the token sequence under construction and $v$ and $u$ denote the currently visited node and the node reached through the traversed edge, respectively. $\mathcal{V}_{\mathrm{vis}}$ and $\mathcal{E}_{\mathrm{vis}}$ represent the sets of visited nodes and edges during sequence construction, respectively. $K$ means maximum number of augmented sequences generated per circuit, $A$ bounds the number of traversal attempts spent to collect them and $T_{\max}$ denotes the fixed maximum sequence length after padding. $\mathrm{ERC}(s)$ denotes an electrical rule check that filters out sequences with invalid connectivity, such as floating internal nets or incomplete device connections. We use $K = 200$, $A = 2000$ and $T_{\max} = 1024$; a circuit therefore contributes at most $K$ sequences, and fewer when its graph admits fewer distinct traversals.

# E Model Architecture and Training Details

## E.1 GPT Decoder

We use a decoder-only Transformer language model for autoregressive circuit generation, following the standard GPT-style architecture [37, 38].

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 $3 \times 10^{-4}$ 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 97,500) 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 3.4) is applied as an external constraint by masking invalid token transitions. Under this setting, the average inference latency is 1.45 s per sample.

## E.2 GAT Classifier

We train a Graph Attention Network (GAT) classifier [39] 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

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

and the normalized attention coefficient is given by

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

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

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 $1 \times 10^{-4}$ and weight decay $10^{-3}$, with dataset ratio weighted cross-entropy loss handling rare class dataset. 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 98.61% on the training set and 91.33% on validation set, with an average inference latency of 2.02 ms per sample.

## E.3 Training Curves and Accuracy

Fig. 8 report the training dynamics of the (a) GPT decoder and the (b),(c) GAT classifier, respectively. Figure (d) illustrates the effect of class-weighted loss employed in our GAT-based circuit type

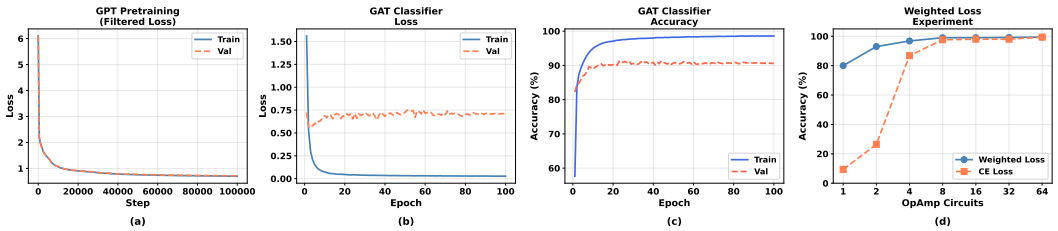



Figure 8: (a) is training curve of GPT model, (b) and (c) are training curve and accuracy of GAT classifier showing lower bound of validation result due to per circuit ID split, (d) shows effectiveness of dataset ratio weighted CE loss result versus conventional CE loss of GAT classifier

classifier. Since our training dataset is inherently imbalanced due to the presence of rare circuit classes, we adopt class-weighted cross-entropy loss, which inversely scales the loss contribution of each class according to its frequency in the training set.

For a controlled evaluation of class imbalance, we construct the dataset as follows. We include 100 samples each from three non-target circuit classes—BGR, LDO, and Power Converter—and split them into training and validation sets with a 9:1 ratio. In addition, we vary the number of OpAmp circuits included in the training set (ranging from 1 to 64) to simulate different levels of data scarcity for the target class. The validation set remains fixed across all settings.

All experiments are conducted using the same GAT architecture and hyperparameter configuration described in the main text, ensuring that performance differences arise solely from the loss formulation. Classification accuracy is evaluated using a held-out set of 500 unseen OpAmp topologies that are not used during GAT training, ensuring a fair assessment of generalization performance.

As shown in the results, class-weighted loss consistently outperforms standard cross-entropy across all data scales, most notably in the extreme few-shot regime where it achieves 80.00% and 93.00% accuracy with only one and two OpAmp training samples, compared to 9.40% and 26.40% for the non-weighted baseline, demonstrating its effectiveness in handling class imbalance under scarce supervision.

# F 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.

## F.1 State Definitions and Decoding Buffers

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.

We define the following six states:

Table 7: State definitions used in the grammar guided decoding process.

| State | Description |
|---|---|
| **Circuit Type – VSS** | Initial decoding state, defined by a circuit-type token followed by the reference ground `VSS`. |
| **Net – Edge** | A state in which a net or external port token is followed by an edge-type token. |
| **Edge – Device** | A state in which an edge-type token is followed by a device token. |
| **Device – Edge** | A state in which a device token is followed by an edge-type token. |
| **Edge – Net** | A state in which an edge-type token is followed by a net or external port token. |
| **Edge – VSS** | A special case of Edge–Net in which the net token corresponds to the reference ground `VSS`. |

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.

Table 8: Decoding buffers in grammar guided decoding

| Buffers | Tracks | Role |
|---|---|---|
| `device-pins` | Which pins of each device have been connected. | All required pins must be used before End is allowed (MOSFET: D/G/S/B; BJT: B/C/E; Diode: P/N). |
| `net_connections` | Which devices are connected to each internal net | Every internal net must connect to $\geq 2$ distinct devices (floating net prevention). |
| `internal_nets_seen` | Internal nets (NET1–NET50) that have appeared in the sequence. | Verify connectivity for all nets that have been generated. |
| `device_edge_nets` | Which net a specific (device, edge) pair is already assigned to. | Prevents the same edge from connecting to two different nets; allows reconnection to the same net |
| `passive_net_count` | Nets connected to each passive device (R/C/L). | A passive is 2-terminal, so it connects to exactly 2 distinct nets. |
| `diode_net_count` | Nets a diode has reached across all of its edges. | A diode is 2-terminal but reaches its terminals through different edges, so two different edges must land on two different nets. |
| `device_pin_nets` | Which nets each individual pin of an active device is assigned to. | Prevents pin-level conflicts for multi-pin edges (e.g., `M_SB` covers both S (Source) and B (Body) pins simultaneously). |

**Decoding buffers.** Decoding buffers and its role in grammar guided decoding is explained in table 8. All buffers are updated incrementally at each decoding step ($O(1)$ per token), avoiding full sequence rescans enabling efficient batch generation.

## F.2 Transition Rules

**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.

### F.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.

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.

### F.4 Retry budget and One-shot validity

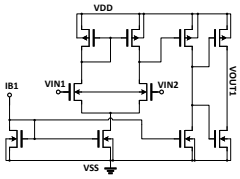
Table 9: Validity under a bounded retry budget. For each of the 15 circuit types, 1,000 samples are generated and each failed sample is regenerated at most $K$ times ($K = 0$: One-shot validity, no regeneration).

| Retry budget | $K=0$ | $K=1$ | $K=2$ | $K=3$ |
|---|---|---|---|---|
| Validity (%) | 72.8 | 87.7 | 92.8 | 95.1 |

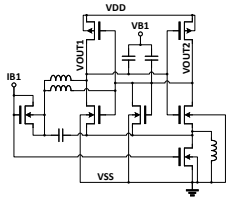
To clarify the termination and regeneration procedure of grammar-guided decoding, we evaluate circuit validity under different retry budgets. A generation attempt is considered successful only when the completion criteria specified in Termination Condition 1 are satisfied within the maximum sequence length. Otherwise, the incomplete sequence is discarded, and generation restarts from

the initial state. Importantly, regeneration is triggered only when the maximum sequence length is reached without satisfying the completion criteria. In our experiments, we observed no cases in which grammar masking eliminated all admissible next tokens, since the grammar always permits valid reconnections to previously used nets or edges.

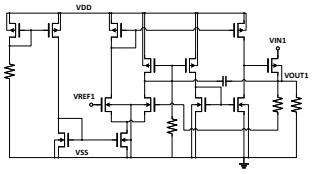
Table 9 reports the validity under different retry budgets. Without regeneration ($K = 0$), one-shot validity is 72.8%. Allowing up to three additional attempts increases validity to 95.1%, demonstrating that regeneration effectively recovers incomplete generation attempts. In our main experiments, the average number of generation attempts per output was 1.49, and the reported validity was evaluated on the final outputs obtained after the regeneration procedure.



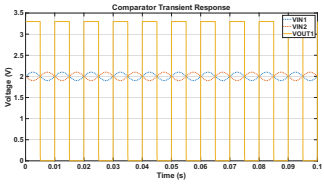

(a) Generated comparator



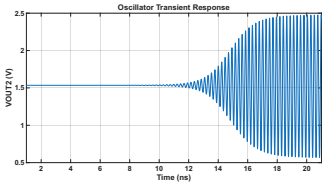

(b) Generated oscillator topology



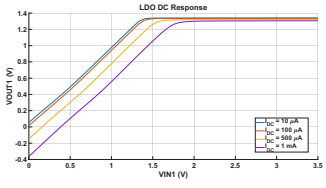

(c) Generated LDO topology




(d) Comparator transient response




(e) Oscillator transient response




(f) LDO DC simulation result across current load

Figure 9: Comparator, oscillator and LDO topologies with their corresponding transient/DC simulation result.

## G Sequence-to-SPICE Translation with default sizing

The generated circuits were translated into netlists with default biasing and sizing, enabling simulation in Cadence Spectre.

For biasing, a supply voltage of VDD = 3.3 V is used. Gate bias voltages are assigned based on device type: 0.75·VDD for PMOS-connected nodes and 0.25·VDD for NMOS-connected nodes. If unspecified, a default bias of 0.5·VDD is applied. For current biasing, a nominal current of 100 $\mu$A is used, with current direction defined as entering NMOS drains and leaving PMOS drains.

For device sizing, NMOS transistors are assigned W/L = 10$\mu$m/1$\mu$m, PMOS transistors W/L = 15$\mu$m/1$\mu$m, resistors R = 10 k$\Omega$, capacitors C = 1 pF, and inductors L = 1 nH.

## H Additional Case Study

### H.1 Comparator Case Study with Transient Simulation

This section presents, for the comparator, oscillator and LDO categories, a representative candidate selected from the simulated netlists in the same way as in Section 4.6. Fig. 9 (a) shows the generated transistor-level comparator circuit. The topology comprises a differential input stage, a pre-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 sequence-to-SPICE decoder with default sizing and biasing logic and simulated under a 45nm CMOS process under Cadence Spectre. The comparator is driven by a $100\,\mathrm{mV}$ peak differential sinusoidal input centered around the common-mode voltage.

Fig. 9 (d) presents the corresponding transient response obtained from simulation. The output exhibits clear comparator behavior and switches decisively between the supply rails. Specifically, the output node transitions fully from 3.3V VDD to VSS, demonstrating robust rail-to-rail swing and confirming correct comparator functionality.

### H.2 Oscillator Case Study with Transient Simulation

Fig. 9 (b) illustrates the generated oscillator topology, which exhibits a cross-coupled structure with an LC tank, consistent with LC oscillator designs. The topology includes differential output nodes (VOUT1 and VOUT2), a biasing network (VB1), and an inductive load, forming a feedback loop that sustains oscillation.

Fig. 9 (e) shows the corresponding transient response obtained from SPICE simulation. The output initially remains near the bias point and then gradually builds up oscillation amplitude due to the inherent startup mechanism driven by noise or small perturbations. As time progresses, the amplitude increases exponentially until it reaches a steady-state limit cycle, indicating that the loop gain exceeds unity during startup and is eventually stabilized by nonlinear effects. The measured oscillation frequency is 4.27 GHz.

### H.3 LDO Case Study with DC Simulation

Fig. 9 (c) illustrates an LDO topology generated by AnalogToBi, consisting of a pass transistor, an error amplifier, and internal biasing circuitry. The output voltage (VOUT1) is fed back through a resistive divider to the error amplifier, where it is compared against the reference voltage (VREF1). The error signal, the amplified gap between the feedback voltage and VREF1, controls the pass device gate, forming a negative feedback loop that regulates VOUT1 against variations in input voltage (VIN1) and load current.

Fig. 9 (f) demonstrates stable output voltage across varying load currents and input voltages. The LDO achieves a load regulation of 10.8 mV/mA and a line regulation of 0.51 mV/V, indicating robust regulation performance. Considering the current-driving capability of the default-sized pass transistor, we perform the simulation with load current sweep to 1 mA.

Table 10: Detailed ablation study results by Circuit Type. These results are obtained under a setting that enables per-type metric evaluation via circuit-type tokens. For each circuit type, 1,000 inference samples are generated.

| | 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 (%) | 88.0 | 93.4 | 79.3 | 84.9 | 68.4 | 84.8 | 84.0 | 91.2 | 86.2 | 90.2 | 95.4 | 87.8 | 85.3 | 84.1 | 89.1 | 86.1 |
| Novelty (%) | 30.4 | 25.3 | 35.7 | 31.1 | 54.0 | 30.0 | 29.3 | 24.2 | 28.3 | 20.3 | 12.5 | 29.8 | 31.6 | 25.0 | 30.4 | 29.2 |
| Valid & Novel (%) | 18.4 | 18.7 | 15.1 | 16.0 | 22.4 | 14.8 | 13.4 | 15.5 | 14.5 | 10.5 | 7.9 | 17.6 | 16.9 | 9.6 | 19.5 | 15.4 |
| Exact. (%) | 56.7 | 58.6 | 53.9 | 53.4 | 37.4 | 57.6 | 56.0 | 61.7 | 59.4 | 63.8 | 72.8 | 56.3 | 60.3 | 59.0 | 54.2 | 57.5 |
| Type Acc. (%) | 96.3 | 89.4 | 94.3 | 99.5 | 99.7 | 95.3 | 83.2 | 97.6 | 96.3 | 90.7 | 99.7 | 93.2 | 99.2 | 90.2 | – | 94.6 |
| **Device-pin Representation + Type Token + Renaming** | | | | | | | | | | | | | | | | |
| Validity (%) | 71.5 | 85.0 | 37.9 | 69.0 | 30.8 | 61.8 | 45.2 | 55.6 | 60.2 | 46.6 | 37.8 | 63.6 | 56.5 | 42.5 | 69.8 | 55.6 |
| Novelty (%) | 75.4 | 67.3 | 90.7 | 86.6 | 90.9 | 81.0 | 94.2 | 75.7 | 93.0 | 87.8 | 88.8 | 82.2 | 75.4 | 90.2 | 82.4 | 84.1 |
| Valid & Novel (%) | 46.9 | 52.3 | 28.6 | 55.6 | 21.7 | 42.8 | 39.4 | 31.3 | 53.2 | 34.4 | 26.6 | 45.8 | 31.9 | 32.7 | 52.2 | 39.7 |
| Exact. (%) | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| Type Acc. (%) | 90.1 | 85.8 | 92.5 | 98.9 | 92.2 | 88.6 | 94.8 | 93.5 | 87.9 | 73.6 | 90.4 | 86.3 | 95.2 | 82.3 | – | 89.4 |
| **Bipartite representation + Type Token** | | | | | | | | | | | | | | | | |
| Validity (%) | 64.2 | 65.0 | 45.4 | 76.0 | 27.7 | 49.8 | 56.4 | 66.1 | 52.2 | 63.2 | 36.6 | 45.7 | 45.8 | 80.1 | 59.2 | 55.6 |
| Novelty (%) | 48.0 | 51.1 | 62.9 | 46.9 | 81.9 | 68.8 | 50.9 | 49.5 | 65.4 | 43.2 | 72.8 | 65.9 | 58.2 | 34.2 | 59.5 | 57.3 |
| Valid & Novel (%) | 13.1 | 16.6 | 9.2 | 23.1 | 10.0 | 19.1 | 8.2 | 16.3 | 18.2 | 7.3 | 10.4 | 12.9 | 5.3 | 14.7 | 19.6 | 13.6 |
| Exact. (%) | 1.5 | 11.4 | 0.3 | 6.0 | 0.0 | 6.5 | 3.0 | 0.0 | 5.7 | 12.2 | 0.0 | 3.9 | 0.0 | 0.0 | 6.2 | 3.8 |
| Type Acc. (%) | 97.8 | 89.5 | 88.2 | 99.9 | 100.0 | 98.0 | 98.7 | 100.0 | 95.5 | 99.2 | 100.0 | 95.3 | 98.6 | 99.9 | – | 97.2 |
| **Bipartite representation + Type Token + Grammar** | | | | | | | | | | | | | | | | |
| 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 |
| Exact. (%) | 1.6 | 3.5 | 0.0 | 1.4 | 0.0 | 4.0 | 2.8 | 0.0 | 0.8 | 2.5 | 0.0 | 2.5 | 0.0 | 0.0 | 2.3 | 1.4 |
| Type Acc. (%) | 95.7 | 83.8 | 87.4 | 99.9 | 99.3 | 97.4 | 97.8 | 99.9 | 91.0 | 96.6 | 100.0 | 88.3 | 92.4 | 99.3 | – | 94.9 |
| **Bipartite representation + Type Token + Renaming + Grammar (Proposed)** | | | | | | | | | | | | | | | | |
| 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** |
| Exact. (%) | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | **0.0** |
| Type Acc. (%) | 95.8 | 92.4 | 86.4 | 99.6 | 100.0 | 96.7 | 93.6 | 100.0 | 95.3 | 98.5 | 100.0 | 89.6 | 95.2 | 95.1 | – | **95.6** |

# I Detailed Ablation Study Results by Circuit Type

Table 10 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 exact sequence matching result than the device–pin representation, indicating reduced memorization at the per-type level.

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.

## NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The abstract and introduction clearly state the main contributions of this work: (1) circuit type conditioning, (2) device renaming augmentation, (3) a bipartite graph representation for analog circuits, (4) grammar-guided decoding. These claims are directly supported by the ablation study and comparison experiments presented in Section 4.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: Section 5 discusses the focus of our study, its limitations, and directions for future research.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: This paper does not include theoretical results, proofs, or formal mathematical analyses. The contributions are primarily algorithmic and empirical in nature.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The implementation details of the dataset and GPT model are described in Section 4. Further details on the GPT model architecture, GAT classifier, and their training curves are provided in Appendix E.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in

some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: We provide the datasets and code at `https://github.com/Seungmin0825/AnalogToBi`.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: Training and test details, including data splits, hyperparameters, and optimizer settings, are specified in Section 4 and Appendix E.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: The paper does not report error bars or confidence intervals. Since evaluation metrics such as Validity and Novelty are computed over 1,000 generated samples per circuit type, the large sample size provides stable estimates, and we leave statistical analysis for future work.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.

- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Compute resources are reported in Appendix E. GPT decoder training takes 22.8 hours and GAT classifier training takes 1.86 hours, both on a single NVIDIA RTX 5880 GPU. The average inference latency is also reported as 1.45 s per sample for the GPT decoder and 2.02 ms per sample for the GAT classifier.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: This research on analog circuit topology generation does not involve human subjects, sensitive data, or any application with potential for misuse. The authors have reviewed the NeurIPS Code of Ethics and confirm that the research conforms to its guidelines in every respect.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [No]

Justification: This work focuses on a methodological contribution to device-level analog circuit topology generation and does not explicitly analyze societal impacts. The scope of the paper is limited to improving structural validity and novelty in generated circuits, rather than studying downstream deployment or real-world usage. While the proposed approach may indirectly contribute to increased efficiency in analog circuit design, assessing its broader societal implications is beyond the scope of this work.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: This work does not involve the release of models or datasets that pose a high risk of misuse. The proposed method focuses on generating device-level analog circuit topologies and does not involve sensitive data, user-generated content, or applications.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: This work uses an existing dataset from AnalogGenie, and the original authors are properly credited through citation. The dataset is publicly available under the MIT License, and the associated paper is released under the CC BY 4.0 license. We respect the terms of use of these assets and use them solely for research purposes.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: We provide new assets at `https://github.com/Seungmin0825/AnalogToBi`.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: This work does not involve crowdsourcing or experiments with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.

- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: This work does not involve any research with human subjects or crowdsourcing. Therefore, IRB approval or equivalent ethical review is not required.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A]

    Justification: LLMs are not used as part of the core methodology. They were used for writing assistance such as editing and proofreading.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.