ECOPILOT: A CIRCUIT-AWARE MULTIMODAL LLM REPRESENTATION LEARNING FRAMEWORK FOR TIMING REASONING AND OPTIMIZATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Digital-circuit timing depends on connectivity, cell behavior, and operating conditions, which are difficult to capture through text alone. We introduce **ECOpilot**, a multimodal representation learning framework that makes these complementary signals available to a pretrained large language model (LLM). A graph encoder and projection align circuit structure with a frozen language backbone through self-supervised masked gate modeling and tool-supervised arrival-time regression. A cell representation combines learned delay-surface features with logical effort, an analytical descriptor of cell driving behavior. Together with textual operating conditions, these representations support timing prediction and tool-guided optimization of cell replacements. On held-out circuits implemented in a TSMC 7 nm technology, ECOpilot achieves 8.6% mean absolute percentage error in arrival-time prediction. Downstream timing engineering change order (ECO) reduces worst-negative-slack magnitude by up to 85.9% and violating-path count by up to 96.0%, measured from per-corner averages across three test designs. Prediction, feature-ablation, and optimization results support combining explicit circuit structure with physical and logical cell knowledge.

1 INTRODUCTION

Digital circuits are built from standard cells, small library-defined circuit blocks connected by wires. Predicting when a signal arrives requires both the connectivity of the circuit and the delay behavior of its cells. That behavior also changes with process, voltage, and temperature (PVT); each PVT setting is called an operating *corner*. A representation for circuit timing must therefore connect topology, local physical behavior, and global operating conditions. We study this problem through timing engineering change order (ECO), which repairs timing violations in an implemented circuit by replacing cells without changing their logic functions. The discrete optimization is NP-hard (Ning, 1994), and a replacement that helps one corner can hurt another.

Learning-based circuit optimization methods predict timing behavior or replacement actions from circuit data (Pham & Chung, 2023; Suhas & Ozev, 2023; Zhang et al., 2024b). Graph neural networks (GNNs) provide a natural representation of connectivity (Lu et al., 2023; Ding et al., 2025; Lu et al., 2025), but topology alone does not describe the physical response of candidate cells. Moreover, learning only from optimization actions can inherit the preferences of the strategy that generated those labels. These limitations motivate learning structural and timing semantics before adapting the representation to action selection.

Large language models (LLMs) offer an interface for combining task instructions, contextual information, and tool feedback. They have been applied to hardware-description generation and verification (Lu et al., 2024; ul Islam et al., 2025; Hu et al., 2025), design-tool scripting (Wu et al., 2024; Chang et al., 2025), design-space exploration (Kochar et al., 2025; Li et al., 2025a), and question answering (Pu et al., 2025; Sharma et al., 2024). However, serializing a circuit into text consumes context and leaves the model to recover connectivity from a sequence (Ho & Ren, 2024; Liu et al., 2025; 2024a). Cell names also omit the delay characteristics needed to compare replacements. This raises our central question: how can circuit structure and cell-level timing knowledge be represented jointly for language-model-based prediction and decision-making?

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

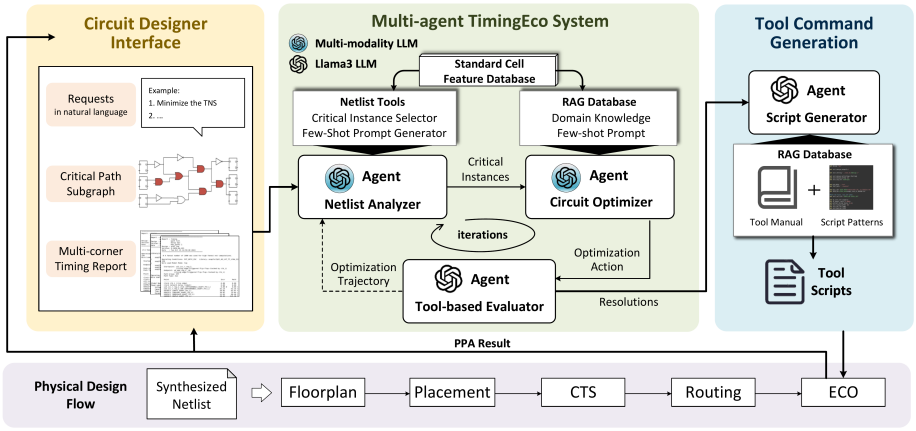

Figure 1: ECOpilot connects multimodal circuit representations to timing reasoning and downstream ECO. Graph structure, cell-level timing knowledge, and corner context inform instance selection and replacement proposals; timing-tool feedback guides search before executable ECO scripts are generated.

ECOpilot addresses this question through two complementary representations. A graph encoder and projection align connectivity with the LLM embedding space using masked gate modeling (MGM) and arrival-time regression (ATR). A cell representation encodes physical delay surfaces and logical effort, while text supplies operating conditions and path context. The model first learns structural and timing semantics, then learns the timing impact of replacements. A tool-guided agentic workflow uses these representations for downstream ECO. The study follows *circuit representation learning* $\rightarrow$ *timing reasoning* $\rightarrow$ *optimization*, with the following contributions.

- **Task-aligned circuit representations.** We jointly train a graph encoder and graph-to-language projection through self-supervised MGM and tool-supervised ATR, aligning structural and timing information with a frozen LLM before action-impact adaptation.
- **Domain-aware cell representations.** We combine learned physical delay features with logical effort to provide compact, complementary cell knowledge for language-model-based timing reasoning.
- **Representation-guided optimization.** We integrate these representations into a tool-guided multi-corner ECO workflow and evaluate their utility through held-out prediction, cell-feature ablations, and implemented circuit optimization.

2 Related Work

Circuit representation learning. DeepGate4 (Zheng et al., 2025) learns circuit representations with a sparse graph transformer and an update strategy designed for large circuit graphs. ECOpilot studies a complementary interface: aligning circuit topology with a pretrained language model while supplying cell behavior and operating conditions for timing tasks. Our evaluation focuses on prediction and replacement decisions rather than large-graph scaling.

Learning-based circuit optimization. Supervised circuit models learn timing quantities or replacement actions from labeled examples (Suhas & Ozev, 2023; Lu et al., 2023). Generative methods such as TransSizer and LEGO-Size (Nath et al., 2023; Lu et al., 2025) broaden the space of proposed solutions. ECOpilot separates circuit pretraining from action-specific adaptation: MGM and ATR learn structural and timing information before action-impact labels are introduced. Corner-conditioned proposals and tool feedback then connect that representation to optimization across operating conditions.

LaMPlace (Geng et al., 2025) connects learned cross-stage metric prediction to macro-placement optimization. Both studies evaluate learned information through downstream design decisions, but

address different tasks: LaMPlace targets placement, whereas ECOpilot proposes cell replacements conditioned on operating corners and evaluates them with timing-tool feedback.

Graph modality with large language models. Graph-to-text methods serialize topology into sequences (Wang et al., 2023), increasing context length and requiring the model to recover structural dependencies from text (Liu et al., 2024b). LLM-enhanced GNNs instead enrich node attributes with semantic features (Amik et al., 2025; Wang et al., 2025; Huang et al., 2025); their message-passing receptive fields can still limit access to distant path dependencies. Multimodal models project graph or image embeddings into a language-model space (Radford et al., 2021; Chai et al., 2023; Tian et al., 2024), and circuit-specific Q-Former models use staged alignment (Li et al., 2025b; 2023b). ECOpilot combines explicit graph embeddings with textual path context and jointly updates the graph encoder and projection through circuit tasks. This allows the structural representation to adapt during alignment, whereas fixing an independently pretrained encoder restricts adaptation to the remaining modules.

Tool-augmented multi-agent systems. Role decomposition separates planning, tool interaction, and code generation (Zhang et al., 2024a; Li et al., 2023a; Hong et al., 2024; Wu et al., 2023), including in design automation and analog-circuit systems (Wu et al., 2025; Liu et al., 2025; 2024a). ECOpilot applies this workflow to standard-cell optimization, using learned circuit representations for proposals and timing-tool feedback for evaluation.

3 METHODOLOGY

3.1 REPRESENTATION AND DOWNSTREAM TASK

Given a circuit graph G after placement and routing and timing reports for corners $k \in \mathcal{K}$, an ECO action $a = (i, c)$ replaces cell instance i with a legal, function-equivalent library cell $c \in \mathcal{C}(i)$. We consider gate sizing, which changes drive strength, and threshold-voltage swapping, which changes the cell's delay–leakage trade-off. The objective is to reduce setup-timing violations. For setup timing, a path's slack is its required arrival time minus its actual arrival time; negative slack indicates a missed timing requirement. We measure worst negative slack (WNS), total negative slack (TNS), and the number of violating paths (#VP), while monitoring power. WNS and TNS improve toward zero, and fewer violating paths are better. Legal candidates and the physical implementation flow determine the admissible replacements.

The model uses two interfaces: graph embeddings are projected into the LLM embedding space, and cell features are retrieved as structured descriptions alongside operating conditions and path context. We use *timing reasoning* to refer to predicting timing quantities and assessing replacements from this combined information.

Figure 1 summarizes the workflow: select candidate instances, propose replacements, evaluate their timing effects, and generate an implementation script. The selection and proposal agents use the multimodal model; evaluation and script generation use the Llama 3.1 backbone (Dubey et al., 2024). We describe circuit alignment, cell representations, and their use in optimization in turn.

3.2 MULTIMODAL CIRCUIT REPRESENTATION LEARNING

3.2.1 MODEL ARCHITECTURE

The model comprises a graph encoder, a trainable projection, and a pretrained LLM (Figure 2). The encoder processes circuit connectivity, and the projection maps its output to the language-model embedding dimension.

We cut the netlist at register boundaries to form a directed acyclic graph of combinational logic, with cell instances as nodes and signal connections as edges. A graph attention network encodes this structure. Its projected output is concatenated with text embeddings and processed by the LLM's self-attention layers.

Text complements the circuit graph with PVT descriptors, retrieved cell features, critical-path descriptions, and optimization instructions. It complements the explicit graph structure with global

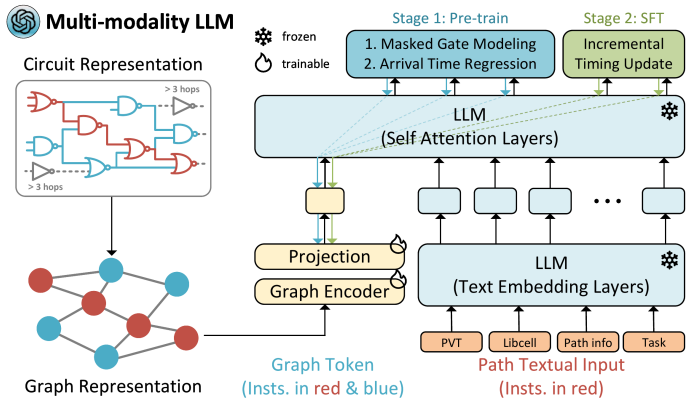

Figure 2: Multimodal circuit representation learning. The graph encoder and projection align circuit structure with a frozen language backbone through MGM and ATR. Action-impact finetuning then connects the learned representation to replacement evaluation, with retrieved cell knowledge and text supplying timing and task context.

operating conditions and task context. Table 5 lists the attributes obtained from the netlist, layout, timing reports, and cell database.

The language backbone remains frozen while the graph encoder and projection are trained. This gives circuit alignment a fixed language representation and is intended to limit interference with pretrained language capabilities under limited circuit data. The interface can be adapted to other decoder-only backbones by changing the projection dimension; all experiments here use Llama 3.1_8B.

3.2.2 Structural and Timing Pretraining

Training proceeds in two stages: circuit pretraining followed by action-impact finetuning. Pretraining combines masked gate modeling (MGM), which reconstructs hidden gate attributes, with arrival-time regression (ATR), which predicts path arrival times from timing-tool labels. MGM uses a reconstruction cross-entropy loss and ATR uses mean absolute error; their weighted sum defines the pretraining objective (Appendix B.1). ATR supplies timing supervision without costly optimization-action labels, reducing the data-generation burden of pretraining. Gradients pass through the frozen LLM to jointly update the graph encoder and projection, allowing alignment to retain task-relevant information and reduce semantic mismatch between independently learned modalities.

3.2.3 Adaptation to Timing Decisions

Stage 2 predicts the tool-measured delay change from an initial critical-path subgraph and a legal replacement action, using mean squared error (Appendix B.2). During optimization, the model proposes legal replacements conditioned on the circuit representation and accumulated timing feedback.

3.3 Domain-Aware Cell Representations

Evaluating a replacement requires knowing how the candidate cell responds to its electrical environment. We represent this behavior using learned physical delay features and an analytical logical descriptor. The resulting compact features are stored in a database and retrieved into the model's task context.

For physical features, a convolutional neural network encodes each cell's two-dimensional timing lookup table (Figure 3). The axes are input slew, the duration of an incoming signal transition, and output load, the capacitance driven by the cell. Table entries specify delay under these conditions. Local convolutional filters provide a way to model curvature, boundary behavior, and sensitivity across this response surface.

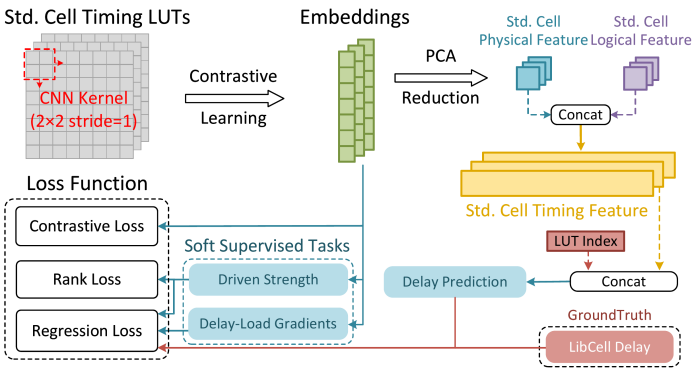

Figure 3: Domain-aware cell representation. Learned physical features describe timing-surface behavior, while logical effort supplies complementary intrinsic cell properties. The combined features are stored for retrieval into the multimodal model's task context.

Contrastive, ranking, and regression objectives train the cell encoder to represent cell identity, relative driving strength, and absolute timing behavior. Principal component analysis compresses the learned features, reducing the prompt space occupied by retrieved cell descriptions. Appendix B gives the objectives and sample construction.

For logical features, logical effort describes driving capability from internal cell topology (Sutherland & Sproull, 1991) through $d = gh + p$, where g is logical effort, h is electrical effort describing relative capacitive load, and p is parasitic delay. This analytical descriptor complements the learned delay-surface features.

We store these representations with area, power, capacitance, function, and legal replacement information. Retrieved features enter the system prompt alongside corner and path descriptions. The cell encoder is trained independently before database construction; circuit pretraining updates the graph encoder and projection through MGM and ATR.

3.4 Representation-Guided Agentic ECO

As shown in Figure 1, the optimization loop separates candidate selection, replacement proposals, timing evaluation, and script generation into four roles. Each proposal receives timing-tool feedback before the final plan is translated into executable commands.

3.4.1 Netlist Analyzer Agent

The Netlist Analyzer identifies critical instances, retrieves their cell features, and assembles few-shot examples. It scores instances on violating paths using corner weights $w_k = \mathrm{softmax}(v_k/\tau)$, where v_k is the timing violation ratio for corner k and τ controls the concentration of the weights.

Algorithm 1 combines each instance's fraction of path delay with the gap between its current and best candidate normalized timing effort. Scores accumulate across corners, and the top-K instances are selected for optimization.

For each selected instance, the analyzer constructs a legal replacement set with the corresponding cell features. It also retrieves same-cell-type optimization examples and their measured timing improvements from a pre-built case database to condition the optimizer.

3.4.2 Circuit Optimizer Agent

The Circuit Optimizer proposes replacements from the candidate sets using the multimodal representation and retrieved examples. If an instance appears on a critical path in only one corner, it selects the action with the highest model output probability. For instances shared across critical paths in multiple corners, it generates corner-specific proposals and combines their probabilities using the weights w_k.

Algorithm 1 Critical Instance Selection

Require: Violating paths P_{vio}, corner weights $\{w_k\}$, number K
Ensure: Top-K critical instances
$I \leftarrow$ all instances on a violating path
for each $i \in I$ **do**
  $s_i \leftarrow 0$
  for each corner k with $w_k > 0$ **do**
    $\text{IMPT}_{i,k} \leftarrow delay_i(k)/path_delay(k)$
    $e^{\text{curr}}_{i,k} \leftarrow normalized_eff(cell_i, k)$
    $e^{\text{best}}_{i,k} \leftarrow \max_{c \in func(i)} normalized_eff(c, k)$
    $s_i \leftarrow s_i + w_k \, \text{IMPT}_{i,k}(e^{\text{best}}_{i,k} - e^{\text{curr}}_{i,k})$
  end for
end for
return Top-K instances ranked by s_i

3.4.3 Tool-Based Evaluator Agent

The Tool-Based Evaluator calls `estimate_eco` for local timing analysis of affected connections. Following the action-feedback pattern of LLM-guided search (Liu et al., 2024c), it appends $(\text{action}_t, \Delta\text{timing}_t)$ pairs to the context for subsequent proposals. Search stops when timing requirements are met or the iteration budget is exhausted. These local estimates guide search; final measurements follow physical implementation.

3.4.4 Script Generator Agent

The Script Generator retrieves tool documentation and script patterns to translate the selected plan into place-and-route commands. Appendix G provides the complete role prompts, command grammars, and examples.

4 Experimental Results and Analysis

We evaluate three questions: whether the circuit representation supports structural recovery and timing prediction, whether physical and logical cell features provide complementary information, and whether the representations support downstream timing optimization. MGM accuracy and ATR error assess reconstruction and prediction, while post-implementation timing metrics assess the resulting ECO decisions. Detailed experimental settings and data acquisition procedures are provided in Appendix A.

4.1 Circuit Representation and Timing Prediction

Table 1 compares ECOpilot with a GNN (Wang et al., 2022), a graph-to-text LLM (Wang et al., 2023), and a Q-Former-based LLM (Li et al., 2025b). MGM accuracy measures masked-attribute recovery; ATR mean absolute percentage error (MAPE) measures arrival-time prediction. The table includes all training and test designs.

ECOpilot obtains a reported test-average MGM accuracy of 97.2% and ATR MAPE of 8.6%, compared with 30.9% and 51.1% for graph-to-text. The advantage appears in both attribute recovery and timing prediction, supporting the use of explicit graph structure for these circuit tasks.

The GNN remains competitive on MGM but has higher ATR error, especially on large circuits. On the largest test design, *tate*, ECOpilot achieves 13.2% MAPE versus 24.5% for the strongest baseline. The different rankings across MGM and ATR highlight that recovering local attributes and predicting path timing are distinct requirements; ECOpilot performs well on both under the evaluated setup.

Table 1: Circuit sizes and representation-learning results. #Insts. and #EPs. denote cell instances and timing endpoints. DAC'22 denotes the GNN baseline (Wang et al., 2022); NIPS'23 denotes graph-to-text (Wang et al., 2023); ICLAD'25 denotes the Q-Former baseline (Li et al., 2025b).

Design	#Insts.	#EPs	**Masked Gate Modeling (Acc.)**				**Arrival Time Regression (MAPE)**			
			DAC'22	NIPS'23	ICLAD'25	ECOpilot	DAC'22	NIPS'23	ICLAD'25	ECOpilot
aes	4570	1320	**99.8%**	97.5%	98.5%	99.1%	6.3%	8.0%	3.5%	**0.7%**
aor3000	175781	128090	96.5%	94.5%	97.8%	**97.8%**	7.9%	9.5%	4.7%	**3.8%**
des	1571	194	**99.5%**	96.0%	98.3%	99.0%	3.5%	6.8%	2.5%	**0.7%**
eth_top	26451	18298	**99.0%**	96.2%	98.4%	97.1%	7.5%	8.3%	4.2%	**1.4%**
nova	129392	57897	98.0%	95.8%	98.2%	**98.2%**	8.2%	11.0%	5.5%	**2.7%**
pci_bridge	8615	2649	98.6%	95.0%	98.0%	**99.4%**	4.8%	7.5%	3.2%	**1.5%**
spi_top	1224	370	**99.8%**	98.0%	98.1%	99.3%	2.0%	4.5%	1.8%	**0.9%**
tv80_core	3877	631	99.1%	93.0%	97.9%	**99.2%**	5.7%	5.2%	3.8%	**1.2%**
Train Avg.	43935	26181	**98.8%**	95.8%	98.2%	98.6%	5.7%	7.6%	3.7%	**1.6%**
ac97_top	5616	3377	**96.8%**	44.3%	92.8%	96.1%	9.9%	43.7%	13.3%	**7.2%**
mc_top	4132	1279	96.6%	42.0%	95.9%	**98.2%**	8.5%	52.2%	8.7%	**4.3%**
tate	211765	53135	94.9%	16.2%	92.0%	**95.6%**	49.6%	63.6%	24.5%	**13.2%**
vga_lcd	37595	32610	95.5%	33.5%	91.5%	**98.5%**	26.8%	37.2%	17.1%	**8.5%**
wb_dma	11322	705	94.0%	18.5%	93.2%	**97.5%**	37.2%	58.8%	18.8%	**9.8%**
Test Avg.	54086	18221	95.3%	30.9%	93.1%	**97.6%**	26.4%	51.1%	16.5%	**8.6%**

Table 2: Standard-cell delay-prediction ablation study of timing features. MAE is reported in ps (lower is better).

Method	Physical	Logical	MAE
Baseline	×	×	3.57 ps
ECOpilot w/o P.F.	×	✓	2.66 ps
ECOpilot w/o L.F.	✓	×	3.09 ps
ECOpilot	✓	✓	**1.36 ps**

4.2 Physical and Logical Knowledge Ablation

We compare the full cell representation with three alternatives (Table 2): conventional function and drive-strength attributes, removal of the learned physical features, and removal of logical effort.

Combining physical and logical features reduces delay-prediction mean absolute error (MAE) from 3.57 to 1.36 ps, a 61.9% reduction. Removing physical or logical features increases MAE to 2.66 or 3.09 ps, respectively. The combined representation outperforms either feature group alone, supporting their complementary value for modeling cell delay.

4.3 Downstream Multi-Corner ECO Evaluation

We evaluate downstream ECO on three held-out designs, *ac97_top*, *vga_lcd*, and *tate*, spanning small, intermediate, and large timing-path counts. Each is evaluated at fast (FF), typical (TT), and slow (SS) operating corners. Tables 3 and 4 report per-design results and per-corner averages for baseline comparisons and representation ablations.

Comparison with baseline methods. Table 3 compares ECOpilot with ECO-GNN (Lu et al., 2023) and a traditional cell-delay-sensitivity baseline (Cell Sensitivity, CS) that greedily ranks candidates by local timing sensitivity using the same timing engine, legal actions, and five-iteration budget. ECOpilot reduces corner-average WNS magnitude by 78.1%–85.9%, TNS magnitude by 97.9%–98.9%, and violating-path count by 82.0%–96.0%. ECO-GNN is competitive at TT, its training corner, but exhibits marked degradation in SS and FF corners. Its corner-specific modeling limits cross-corner generalization, leading to incomplete timing repair outside training conditions. CS achieves comparable WNS, with individual cases favoring either method. Its TNS magnitude and violating-path count exceed ECOpilot's by 17.65% and 24.25%, respectively, averaged over the nine design–corner pairs using ECOpilot as the denominator. A possible explanation is that local improvements interact through shared cells, fanout branches, and paths that diverge and later re-

Table 3: Performance comparison with Cell Sensitivity (CS) and ECO-GNN (Lu et al., 2023) across circuit scales and PVT corners. WNS and TNS are better closer to zero; fewer violating paths are better. Parenthesized values in the average rows report improvements relative to the initial status.

Design (#Inst.)	Corner	Initial Status				Cell Sensitivity				ECO-GNN				ECOpilot			
		WNS (ps)	TNS (ns)	#VP	Power (mW)	WNS (ps)	TNS (ns)	#VP	Power (mW)	WNS (ps)	TNS (ns)	#VP	Power (mW)	WNS (ps)	TNS (ns)	#VP	Power (mW)
ac97_top (6k)	FF	-93	-8.279	264	13.92	-12	-0.042	4	13.98	-18	-0.051	3	13.94	-12	-0.036	3	13.97
	TT	-137	-13.156	275	10.61	-18	-0.053	3	10.67	-19	-0.055	3	10.66	-17	-0.051	3	10.66
	SS	-277	-31.196	371	5.82	-45	-0.492	42	5.95	-69	-0.565	42	5.94	-49	-0.418	40	5.94
vga_lcd (38k)	FF	-67	-7.723	363	115.54	-10	-0.221	37	115.61	-19	-0.678	126	115.60	-12	-0.183	28	115.60
	TT	-86	-11.148	366	74.85	-10	-0.044	13	74.89	-12	-0.102	31	74.88	-10	-0.042	7	74.89
	SS	-141	-21.493	430	36.81	-14	-0.147	25	36.83	-37	-0.950	63	36.84	-14	-0.08	16	36.83
tate (212k)	FF	-51	-8.195	84	259.59	-22	-0.127	14	259.75	-29	-0.337	18	259.70	-22	-0.119	14	259.70
	TT	-100	-4.936	104	193.24	-37	-0.219	21	193.34	-36	-0.263	31	193.30	-37	-0.217	20	193.30
	SS	-176	-2.364	214	105.17	-21	-0.680	128	105.21	-53	-1.244	140	105.20	-21	-0.656	127	105.20
Avg. (impr.)	FF	-70	-8.066	237	129.68	-14.7 (79.0%)	-0.130 (98.4%)	18.3 (92.3%)	129.78 (-0.08%)	-22 (68.6%)	-0.355 (95.6%)	49 (79.3%)	129.75 (-0.05%)	-15.3 (78.1%)	-0.113 (98.6%)	15 (93.7%)	129.76 (-0.06%)
	TT	-108	-9.747	248	92.9	-21.7 (79.9%)	-0.105 (98.9%)	12.3 (95.0%)	92.97 (-0.07%)	-22.3 (79.3%)	-0.140 (98.6%)	22 (91.3%)	92.95 (-0.05%)	-21.3 (80.3%)	-0.103 (98.9%)	10 (96.0%)	92.95 (-0.05%)
	SS	-198	-18.351	338	49.27	-26.7 (86.5%)	-0.440 (97.6%)	65.0 (80.8%)	49.33 (-0.12%)	-53.0 (73.2%)	-0.920 (94.9%)	82 (75.8%)	49.33 (-0.12%)	-28 (85.9%)	-0.385 (97.9%)	61 (82.0%)	49.32 (-0.10%)

Table 4: Ablation study of the graph encoder and standard-cell feature database across circuit scales and PVT corners. WNS and TNS are better closer to zero; fewer violating paths are better. Parenthesized values in the average rows report improvements relative to the initial status.

Design (#Inst.)	Corner	Initial Status				ECOpilot w/o Graph Encoder				ECOpilot w/o Feature Database				ECOpilot (Full)			
		WNS (ps)	TNS (ns)	#VP	Power (mW)	WNS (ps)	TNS (ns)	#VP	Power (mW)	WNS (ps)	TNS (ns)	#VP	Power (mW)	WNS (ps)	TNS (ns)	#VP	Power (mW)
ac97_top (6k)	FF	-93	-8.279	264	13.92	-33	-0.316	21	13.94	-14	-0.041	3	13.94	-12	-0.036	3	13.97
	TT	-137	-13.156	275	10.61	-44	-0.501	25	10.65	-19	-0.057	3	10.65	-17	-0.051	3	10.66
	SS	-277	-31.196	371	5.82	-96	-5.372	89	5.93	-52	-0.549	47	5.93	-49	-0.418	40	5.94
vga_lcd (38k)	FF	-67	-7.723	363	115.54	-27	-1.445	162	115.61	-11	-0.302	85	115.67	-12	-0.183	28	115.60
	TT	-86	-11.148	366	74.85	-31	-1.742	141	74.89	-13	-0.319	73	74.93	-10	-0.042	7	74.89
	SS	-141	-21.493	430	36.81	-53	-1.871	89	36.82	-16	-0.216	49	36.85	-14	-0.08	16	36.83
tate (212k)	FF	-51	-8.195	84	259.59	-38	-0.419	21	259.71	-22	-0.123	14	259.80	-22	-0.119	14	259.70
	TT	-100	-4.936	104	193.24	-75	-1.018	49	193.30	-37	-0.235	25	193.35	-37	-0.217	20	193.30
	SS	-176	-2.364	214	105.17	-59	-1.75	178	105.20	-31	-0.779	133	105.22	-21	-0.656	127	105.20
Avg. (impr.)	FF	-70	-8.066	237	129.68	-32.7 (53.3%)	-0.727 (90.9%)	68 (71.3%)	129.75 (-0.05%)	-15.7 (77.6%)	-0.155 (98.1%)	34 (85.7%)	129.8 (-0.09%)	-15.3 (78.1%)	-0.113 (98.6%)	15 (93.7%)	129.76 (-0.06%)
	TT	-108	-9.747	248	92.9	-50 (53.7%)	-1.087 (88.9%)	72 (70.9%)	92.95 (-0.05%)	-23 (78.7%)	-0.204 (97.9%)	34 (86.3%)	92.98 (-0.09%)	-21.3 (80.3%)	-0.103 (98.9%)	10 (96.0%)	92.95 (-0.05%)
	SS	-198	-18.351	338	49.27	-69.3 (65.0%)	-2.998 (83.7%)	119 (64.8%)	49.32 (-0.10%)	-33 (83.3%)	-0.515 (97.2%)	76 (77.5%)	49.33 (-0.12%)	-28 (85.9%)	-0.385 (97.9%)	61 (82.0%)	49.32 (-0.10%)

converge; initially sub-critical paths can also become limiting. Graph-conditioned proposals and iterative feedback provide a mechanism for accounting for these effects.

Ablation of circuit-representation components. Removing the graph encoder reduces mean WNS improvement from 81.4% to 57.3%, averaging the three reported corner-level improvement percentages equally (Table 4). TNS and violating-path count also worsen. Removing retrieved cell features retains stronger WNS recovery, but produces higher aggregate TNS magnitude and more violating paths than the full model in every corner. Its rounded TT aggregate power overhead is 0.09%, versus 0.05% for the full model: approximately $1.8\times$ for this aggregate. The ablations support complementary roles for connectivity and cell-level knowledge: the graph describes where replacements interact, while cell features describe candidate delay behavior.

4.4 Workflow Reliability and Stability

To evaluate role decomposition, we combine all role instructions into one prompt for a single-agent baseline using the same models, tools, and legal actions. Over 50 executions per stage and 10 complete runs, ECOpilot improves action legality from 84% to 98%, tool invocation from 72% to 98%, script validation from 82% to 100%, and end-to-end completion from 60% to 100%. Appendix C defines success and describes failure modes.

We measure run-to-run variability over five runs on *tate*, using deterministic decoding for analysis and command generation and sampled decoding for action proposals. The maximum normalized standard deviation $\sigma/|\mu|$ across corners is 5.35% for WNS and 1.98% for violating-path count (Appendix D).

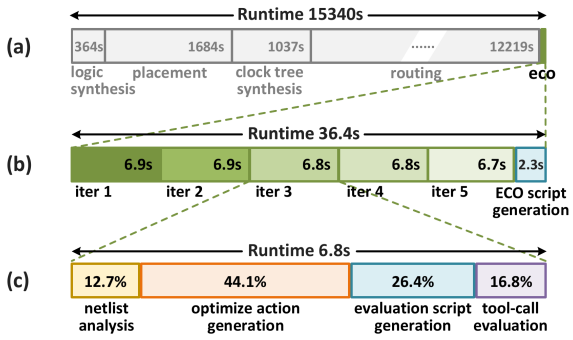

Figure 4: Runtime analysis on *tate*: (a) the reported design-flow overview, (b) five-iteration search and final script generation, and (c) the third-iteration breakdown. The plotted ECO segment denotes search and script generation; the separately measured 188 s physical ECO implementation is excluded.

4.5 Training Cost and Optimization Runtime

Training cost. At batch size 8, LoRA with rank $r = 8$ and $\alpha = 16$ improves the reported MGM accuracy from 97.2% to 98.1% and ATR MAPE from 8.6% to 8.3% (Table 7). GPU memory rises from 19.1 to 34.6 GB and training time from 1161 to 1270 minutes. Updating few parameters still requires intermediate activations for backpropagation through adapted layers. The limited accuracy gain relative to these costs motivates using the frozen backbone by default.

Optimization runtime. ECOpilot uses local timing feedback during search and physically implements the final plan once. We use five iterations based on the diminishing returns observed in the budget study (Appendix E). On *tate*, five iterations take 6.7–6.9 s each, followed by 2.3 s for script generation, totaling 36.4 s (Figure 4). In the third iteration, netlist analysis accounts for 12.7%, action generation for 44.1%, evaluation-script generation for 26.4%, and timing-tool evaluation for 16.8%. Action and script generation therefore dominate the measured search cost. Physical ECO implementation takes a further 188 s, outside the plotted search stage.

5 Conclusion

We presented **ECOpilot**, a multimodal representation learning framework for circuit timing tasks. It aligns graph structure with a frozen LLM and supplies complementary physical and logical cell knowledge as context. Held-out prediction, cell-feature ablations, and downstream ECO support the utility of these representations for timing estimation and replacement decisions. Evaluation is limited to the tested technology and operating corners; broader transfer on different technology remain future work (Appendix F).

AI USE STATEMENT

Generative AI tools were used to assist language refinement during the preparation of this manuscript, including grammar refinement and readability improvement. All literature reviews, technical contributions, experiments, analyzes, and conclusions were independently developed and verified by the authors.

REPRODUCIBILITY STATEMENT

Appendix A describes the hardware, circuit-level train/test split, and distinction between local timing estimates and implemented signoff results. Appendix B provides training objectives and available configuration details; Table 1 provides benchmark-level representation results, and Appendices C–D provide reliability and stability evaluation protocols; Appendix G provides the agent prompt templates. Reproduction requires access to the commercial technology library and EDA tools; these access limitations are discussed in Appendix F.

REFERENCES

Fahad Rahman Amik, Yousef Safari, Zhanguang Zhang, and Boris Vaisband. Graph-based timing prediction at early-stage rtl using large language model. In *Proceedings of the 30th Asia and South Pacific Design Automation Conference*, pp. 393–400, 2025.

Ziwei Chai, Tianjie Zhang, Liang Wu, Kaiqiao Han, Xiaohai Hu, Xuanwen Huang, and Yang Yang. Graphllm: Boosting graph reasoning ability of large language model. In *arXiv preprint*, 2023.

Chen-Chia Chang, Chia-Tung Ho, Yaguang Li, Yiran Chen, and Haoxing Ren. Drc-coder: Automated drc checker code generation using llm autonomous agent. In *Proceedings of the 2025 International Symposium on Physical Design (ISPD)*, pp. 143–151, 2025.

Wenjie Ding, Zhanhua Zhang, Guoqing He, and Peng Cao. A physical and timing aware placement optimization framework based on graph neural network. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2025.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv e-prints*, pp. 1–92, 2024.

Zijie Geng, Jie Wang, Ziyan Liu, Siyuan Xu, Zhentao Tang, Shixiong Kai, Mingxuan Yuan, Jianye Hao, and Feng Wu. LaMPlace: Learning to optimize cross-stage metrics in macro placement. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/04c0399a47ee4107cd03b08f1f8c3eeb-Abstract-Conference.html.

Chia-Tung Ho and Haoxing Ren. Large language model (llm) for standard cell layout design optimization. In *2024 IEEE LLM Aided Design Workshop (LAD)*, pp. 1–6, 2024.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Zhang, Ceyao Wang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. Metagpt: Meta programming for multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*, 2024.

Yuchen Hu, Junhao Ye, Ke Xu, Jialin Sun, Shiyue Zhang, Xinyao Jiao, Dingrong Pan, Jie Zhou, Ning Wang, Weiwei Shan, Xinwei Fang, Xi Wang, Nan Guan, and Zhe Jiang. Uvllm: An automated universal rtl verification framework using llms. In *Proceedings of the 62nd ACM/IEEE Design Automation Conference (DAC)*, pp. 1–7, 2025.

Shan Huang, Jinhao Li, Zhen Yu, Jiancai Ye, Jiaming Xu, Ningyi Xu, and Guohao Dai. Llsm: Llm-enhanced logic synthesis model with eda-guided cot prompting, hybrid embedding and aig-tailored acceleration. In *Proceedings of the 30th Asia and South Pacific Design Automation Conference*, pp. 974–980, 2025.

Dimple Vijay Kochar, Hanrui Wang, Anantha P. Chandrakasan, and Xin Zhang. Ledro: Llm-enhanced design space reduction and optimization for analog circuits. In *2025 IEEE International Conference on LLM-Aided Design (ICLAD)*, pp. 141–148, 2025.

Guohao Li, Hasan Abed Al Kader Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for mind exploration of large scale language model society. *arXiv preprint arXiv:2303.17760*, 2023a.

Jingyuan Li, Jianrong Zhang, Ye Li, Wenbo Yin, and Lingli Wang. Lemoe: Llm-enhanced multi-objective bayesian optimization for microarchitecture exploration. In *Proceedings of the 62nd ACM/IEEE Design Automation Conference (DAC)*, pp. 1–7, 2025a.

Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *Proceedings of the 40th International Conference on Machine Learning*, 2023b.

Wei Li, Yang Zou, Christopher Ellis, Ruben Purdy, Shawn Blanton, and José M. F. Moura. Bridges: Bridging graph modality and large language models within eda tasks. In *2025 IEEE International Conference on LLM-Aided Design (ICLAD)*, pp. 77–84, 2025b.

Bingyang Liu, Haoyi Zhang, Xiaohan Gao, Zichen Kong, Xiyuan Tang, Yibo Lin, Runsheng Wang, and Ru Huang. Layoutcopilot: An llm-powered multiagent collaborative framework for interactive analog layout design. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 44(8):3126–3139, 2025.

Chengjie Liu, Weiyu Chen, Anlan Peng, Yuan Du, Li Du, and Jun Yang. Ampagent: An llm-based multi-agent system for multi-stage amplifier schematic design from literature for process and performance porting. In *arXiv preprint*, 2024a.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024b.

Tennison Liu, Nicolás Astorga, Nabeel Seedat, and Mihaela van der Schaar. Large language models to enhance bayesian optimization. In *The Twelfth International Conference on Learning Representations*, 2024c.

Yao Lu, Shang Liu, Qijun Zhang, and Zhiyao Xie. Rtllm: An open-source benchmark for design rtl generation with large language model. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 722–727, 2024.

Yi-Chen Lu, Siddhartha Nath, Sai Pentapati, and Sung Kyu Lim. Eco-gnn: Signoff power prediction using graph neural networks with subgraph approximation. *ACM Transactions on Design Automation of Electronic Systems*, 28(4), 2023.

Yi-Chen Lu, Kishor Kunal, Geraldo Pradipta, Rongjian Liang, Ravikishore Gandikota, and Haoxing Ren. Lego-size: Llm-enhanced gpu-optimized signoff-accurate differentiable vlsi gate sizing in advanced nodes. In *Proceedings of the 2025 International Symposium on Physical Design (ISPD)*, pp. 152–162, 2025.

Siddhartha Nath, Geraldo Pradipta, Corey Hu, Tian Yang, Brucek Khailany, and Haoxing Ren. Transsizer: A novel transformer-based fast gate sizer. In *Proceedings of the 41rd IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–9, 2023.

Wing Ning. Strongly np-hard discrete gate-sizing problems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 1994.

OpenCores. Opencores: Open source ip-cores. `http://www.opencores.org`, 2026.

Phuoc Pham and Jaeyong Chung. Agd: A learning-based optimization framework for eda and its application to gate sizing. In *Proceedings of the 60th ACM/IEEE Design Automation Conference*, pp. 1–6, 2023.

Yuan Pu, Zhuolun He, Tairu Qiu, Haoyuan Wu, and Bei Yu. Customized retrieval augmented generation and benchmarking for eda tool documentation qa. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2025.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pp. 8748–8763, 2021.

Utsav Sharma, Bing-Yue Wu, Sai Rahul Dhanvi Kankipati, Vidya A. Chhabria, and Austin Rovinski. Openroad-assistant: An open-source large language model for physical design tasks. In *Proceedings of the 2024 ACM/IEEE International Symposium on Machine Learning for CAD (MLCAD)*, 2024.

Krishna Kashyap Suhas and Sule Ozev. Improved: Integrated method to predict post-routing setup violations in early design stages. *ACM Transactions on Design Automation of Electronic Systems*, 28, 2023. doi: 10.1145/3572546.

Ivan E. Sutherland and Robert F. Sproull. Logical effort: designing for speed on the back of an envelope. In *Proceedings of the 1991 University of California/Santa Cruz Conference on Advanced Research in VLSI*, 1991.

Yijun Tian, Huan Song, Zichen Wang, Haozhu Wang, Ziqing Hu, Fang Wang, Nitesh V Chawla, and Panpan Xu. Graph neural prompting with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19080–19088, 2024.

Mubashir ul Islam, Humza Sami, Pierre-Emmanuel Gaillardon, and Valerio Tenace. Eda-aware rtl generation with large language models. In *2025 Design, Automation & Test in Europe Conference (DATE)*, pp. 1–6, 2025.

Heng Wang, Shangbin Feng, Tianxing He, Zhaoxuan Tan, Xiaochuang Han, and Yulia Tsvetkov. Can language models solve graph problems in natural language? In *Proceedings of the 37th International Conference on Neural Information Processing Systems (NIPS)*, 2023.

Mingjun Wang, Bin Sun, Jianan Mu, Feng Gu, Boyu Han, Tianmeng Yang, Xinyu Zhang, Silin Liu, Yihan Wen, Hui Wang, Jun Gao, Zhiteng Chao, Husheng Han, Zizhen Liu, Shengwen Liang, Jing Ye, Bei Yu, Xiaowei Li, and Huawei Li. Moss: Multi-modal representation learning on sequential circuits. In *Proceedings of the 62nd ACM/IEEE Design Automation Conference (DAC)*, pp. 1–7, 2025.

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

Haoyuan Wu, Zhuolun He, Xinyun Zhang, Xufeng Yao, Su Zheng, Haisheng Zheng, and Bei Yu. Chateda: A large language model powered autonomous agent for eda. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 43(10):3184–3197, 2024.

Haoyuan Wu, Haisheng Zheng, Zhuolun He, and Bei Yu. Divergent thoughts toward one goal: Llm-based multi-agent collaboration system for electronic design automation. In *Annual Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics*, 2025.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. Autogen: Enabling next-gen llm applications via multi-agent conversation. *arXiv preprint arXiv:2308.08155*, 2023.

Ceyao Zhang, Kaijie Yang, Siyi Hu, Zihao Wang, Guanghe Li, Yihang Sun, Cheng Zhang, Zhaowei Zhang, Anji Liu, Song-Chun Zhu, Xiaojun Chang, and Yixin Zhang. Proagent: Building proactive cooperative agents with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 17591–17599, 2024a.

Zhanhua Zhang, Wenjing Ding, Guoqing He, and Peng Cao. Lag-sizer: A novel gate sizer based on leak generative adversarial network with feature fusion. In *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 1–9, 2024b.

Ziyang Zheng, Shan Huang, Jianyuan Zhong, Zhengyuan Shi, Guohao Dai, Ningyi Xu, and Qiang Xu. DeepGate4: Efficient and effective representation learning for circuit design at scale. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/d75f561eaaf2cb754bc8d7e36d8af362-Abstract-Conference.html.

A Experimental Setup and Reproducibility

We implement ECOpilot in PyTorch and PyG with Llama 3.1_8B and a four-layer graph attention network with eight heads per layer and hidden dimension 2048. Training uses batch size 8, learning rate 1×10^{-5}, weight decay 0.05, and three epochs of linear warmup, for up to ten epochs with early stopping. Runs use two NVIDIA A800 GPUs and Intel Xeon Gold 6348 CPUs and complete within one day.

The OpenCores benchmarks (OpenCores, 2026) comprise eight training circuits and five held-out test circuits, ranging from approximately 1 K cell instances (*spi_top*) to 212 K (*tate*). We extract more than 300K critical-path subgraphs with three-hop neighborhoods from routed netlists and physical-layout files. Subgraphs from the eight training circuits are split 80:20 for training and validation; all subgraphs from the five test circuits are excluded from training.

Synopsys Design Compiler synthesizes the circuits, and Cadence Innovus places and routes them using a TSMC 7 nm library. Synopsys PrimeTime supplies static timing analysis (STA) labels and final multi-corner measurements. We distinguish fast local estimates during search from *signoff* measurements, the final timing checks after the replacement script has been physically implemented. All reported ECO timing and power values use the implemented designs. Appendices B–D provide configurations, detailed results, and evaluation protocols.

B Circuit and Cell Representation Learning Details

We provide the circuit-pretraining, action-impact, and cell-feature objectives, followed by the training and inference configuration. The cell encoder is trained before database construction; MGM and ATR train the graph-to-language interface.

B.1 Representation Pretraining

Table 5 lists the graph and text inputs. Nodes correspond to cell instances and directed edges to signal connections; text supplies the operating conditions and task context.

Table 5: Input attributes of the multimodal LLM. Graph attributes represent circuit structure and local instance properties; text provides operating conditions, path context, cell knowledge, and the requested task.

Modality	Type	Description
Graph	Function	Function type
	Fanout	Number of driven instances
	Location	Instance location (x, y)
	Pin location	Output-pin location (x, y)
	Load	Output-net capacitance
	Distance	Driver-to-load distance
	Mean delay	Mean cell delay
Text	PVT	Process, voltage, temperature
	LibCell	Standard-cell features
	Path	Critical-path information
	Task	Human optimization request

Masked gate modeling (MGM) randomly masks gate attributes such as function, drive-strength class, and threshold-voltage class. The model reconstructs the hidden attributes from graph context and text:

$$\mathcal{L}_{\mathrm{mgm}} = -\sum_{i \in \mathcal{M}} \log p_\theta(x_i \mid G_{\setminus \mathcal{M}}, X_{\mathrm{text}}). \tag{1}$$

Arrival-time regression (ATR) predicts a path arrival time $\hat{a}_p$ from the aligned graph-text representation:

$$\mathcal{L}_{\mathrm{atr}} = \frac{1}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} |\hat{a}_p - a_p|. \tag{2}$$

The pretraining objective is

$$\mathcal{L}_{\text{pre}} = \lambda_{\text{mgm}}\mathcal{L}_{\text{mgm}} + \lambda_{\text{atr}}\mathcal{L}_{\text{atr}}. \tag{3}$$

Here, $\mathcal{M}$ is the masked-node set, x_i is the target attribute of node i, $G_{\setminus\mathcal{M}}$ is the graph with these attributes masked, and X_{text} is the textual context. The model with trainable parameters θ assigns reconstruction probability p_θ. The set $\mathcal{P}$ contains the supervised paths, with predicted arrival time $\hat{a}_p$ and tool-generated target a_p. The coefficients λ_{mgm} and λ_{atr} weight the two objectives.

B.2 Supervised Action-Impact Finetuning

Each example pairs a critical-path subgraph and legal replacement action with the resulting tool-measured delay change. The action specifies an instance and a function-equivalent replacement cell. The regression objective is

$$\mathcal{L}_{\text{act}} = \frac{1}{N}\sum_{n=1}^{N}\left\|\widehat{\Delta t}_n - \Delta t_n\right\|_2^2. \tag{4}$$

Here, N is the number of supervised examples, Δt_n is the tool-measured delay change for example n, and $\widehat{\Delta t}_n$ is its prediction.

B.3 Standard-Cell Physical Features

A convolutional encoder processes the two-dimensional timing lookup table indexed by input slew and output load. Data augmentation includes:

- bounded perturbations of LUT values;
- interpolation and sampling perturbations;
- small index jitter along slew and load axes.

Positive pairs are augmented views of the same cell; negative pairs use different cells. The contrastive objective encourages separation between their representations:

$$\mathcal{L}_{\text{con}} = -\log\frac{\exp(\text{sim}(\mathbf{z}_i, \mathbf{z}_i^+)/\tau)}{\sum_{j\in\mathcal{N}(i)}\exp(\text{sim}(\mathbf{z}_i, \mathbf{z}_j)/\tau)}. \tag{5}$$

Here, $\mathbf{z}_i$ is the anchor embedding, $\mathbf{z}_i^+$ is the embedding of another augmented view of the same cell, and $\mathbf{z}_j$ is the embedding of a negative cell. $\mathcal{N}(i)$ denotes the set of negative samples for anchor i, $\text{sim}(\cdot,\cdot)$ is cosine similarity, and τ is the contrastive temperature.

The ranking objective encourages predictions to respect relative driving strength:

$$\mathcal{L}_{\text{rank}} = \sum_{(i,j)}\max(0, m - (y_i - y_j)(\hat{y}_i - \hat{y}_j)), \tag{6}$$

where (i,j) ranges over pairs of driving strength samples, y_i and y_j are their ground-truth driving strength ranks, $\hat{y}_i$ and $\hat{y}_j$ are the corresponding predictions, and m is the ranking margin.

The regression objective fits the absolute target values:

$$\mathcal{L}_{\text{reg}} = \|\hat{\mathbf{y}} - \mathbf{y}\|_2^2. \tag{7}$$

Here, $\hat{\mathbf{y}}$ and $\mathbf{y}$ collect the predicted and ground-truth timing behavior values, including driving strength and delay-load gradients, respectively, and $\|\cdot\|_2$ denotes the Euclidean norm. We use contrastive temperature $\tau = 0.1$ and ranking margin $m = 0$. Principal component analysis compresses the learned embedding before it is stored in the retrieval database.

B.4 Training Configuration

Table 6: Training and inference configuration.

Item	Configuration
Backbone	Llama 3.1_8B
Graph encoder	4-layer GAT
Attention heads	8 per layer
Hidden dimension	2048
Batch size	8
Maximum epochs	10 with early stopping
Learning rate	1×10^{-5}
Weight decay	0.05
Warmup	First 3 epochs
Training hardware	2 NVIDIA A800 GPUs
Analyzer decoding	Deterministic
Optimizer decoding	Temperature 0.3, top-p 0.8
Evaluator decoding	Deterministic
Script decoding	Deterministic

The analyzer, evaluator, and script generator use deterministic decoding. The optimizer samples proposals with temperature 0.3 and top-p 0.8; proposals are restricted to legal candidates and evaluated by the timing tool before implementation.

Table 7: Frozen-backbone and LoRA comparison at batch size 8, with LoRA rank $r = 8$ and $\alpha = 16$. MGM accuracy is higher-is-better; ATR error, memory, and training time are lower-is-better.

Configuration	MGM Acc. (%)	ATR MAPE (%)	GPU memory (GB)	Training time (min)
Frozen LLM	97.2	8.6	19.1	1161
LoRA	98.1	8.3	34.6	1270

C Single-Agent Reliability Evaluation

The single-agent baseline combines all role instructions in one prompt and uses the same models, tools, and legal actions. A stage-level run succeeds if it finishes with parseable, legal output. A complete run succeeds if it finishes, produces an executable script, and improves timing.

Task definitions. **Optimization action generation** checks whether the generated ECO actions satisfy instance- and cell-legality constraints. **Tool-call invocation** checks whether the requested tool execution finishes successfully and returns parseable feedback. **ECO-script validation** checks whether the generated ECO commands pass syntax and object-reference validation. **End-to-end ECO completion** checks whether a complete timing ECO run finishes successfully and achieves a timing improvement.

Table 8: Reliability over 50 stage-level executions and 10 complete runs.

Task	Single agent	ECOpilot
Optimization action generation	84%	98%
Tool-call invocation	72%	98%
ECO-script validation	82%	100%
End-to-end ECO completion	60%	100%

The largest success-rate gap occurs in complete runs, where errors from multiple stages can accumulate. This comparison evaluates the combined effect of role separation and prompt organization.

Common single-agent failures include mixing tool-command dialects, adding text that breaks parsers, selecting cells outside the legal set, and altering instance names after timing feedback is appended to the context.

D Repeated-Run Stability

Table 9: Five repeated runs on *tate* with five optimization iterations.

Metric	Corner	Repeated Runs					μ	σ	$\sigma/\lvert\mu\rvert$
		1	2	3	4	5			
WNS (ps)	FF	-22	-22	-23	-22	-22	-22.2	0.4	0.018018
	TT	-37	-37	-36	-37	-38	-37	0.632456	0.017093
	SS	-21	-21	-24	-21	-22	-21.8	1.16619	0.053495
#VP	FF	14	14	14	14	14	14	0	0
	TT	20	20	21	20	20	20.2	0.4	0.019802
	SS	127	127	129	127	127	127.4	0.8	0.006279

Across corners, the maximum normalized standard deviation $\sigma/|\mu|$ is 5.35% for WNS and 1.98% for violating-path count. These five runs produce similar signoff outcomes under the specified decoding and validation settings.

E Optimization Iterations and Search Budget

In order to evaluate how the number of optimization iterations affects ECOpilot's performance and determine an appropriate iteration setting, we compare the optimization results of five repeated runs on *tate* with one, five, and ten iterations.

It shows that most timing recovery occurs at an early stage: after one iteration, WNS improves from -100 to -52 ps and TNS reduces from -4.94 to -1.06 ns, corresponding to 52% and 21.5% of their initial values, respectively. Five iterations provide further improvements, achieving WNS of -37 ps and TNS of -0.22 ns which are 37% and 4.5% of the initial values. In contrast, ten iterations yield only marginal gains, where WNS remains unchanged and TNS is slightly reduced to -0.19 ns which is only 0.6% improvement. This diminishing return is expected, as timing ECO is a late-stage local optimization, where the remaining violating paths are increasingly constrained by placement, routing congestion and design rules. Therefore, the subsequent performance gain is bounded by the intrinsic limitations. Due to this, ECOpilot adopts five iterations as the default setting to balance optimization quality and runtime.

F Discussion and Limitations

The results support combining explicit topology, cell behavior, and operating context for circuit timing tasks. They establish the utility of the evaluated model, while leaving several mechanisms unresolved. Optimization covers gate sizing and threshold-voltage swapping at the tested corners; unseen PVT conditions, other technologies remain untested.

- **Technology access.** The TSMC 7 nm library cannot be redistributed. Reproduction on another library requires regenerating cell features and timing labels.
- **Tool dependence.** The prototype uses commercial placement, routing, and timing-analysis commands. Reproduction with open tools requires adapting these interfaces.
- **Action space.** We evaluate gate sizing and threshold-voltage swapping. Adding buffers, duplicating cells, and modifying routing are outside the tested action space.

G Complete Agent Prompt Templates

The following templates specify the four roles used in the optimization workflow, including their expected inputs, output formats, and tool commands.

G.1 Netlist Analyzer Agent

Role. You are the Netlist Analyzer Agent in ECOpilot. Your task is to identify timing-critical instances from multi-corner violating paths and provide the selected instance list for downstream timing ECO optimization.

Inputs.

- `designer_request`: Natural-language timing ECO objective, e.g., "minimize TNS" or "optimize critical path delay".
- `violating_paths`: Violating timing paths extracted from multi-corner timing reports.
- `multi_corner_timing_reports`: Timing reports under multiple PVT corners.
- `K`: Number of critical instances to select.

Available External Tool.

- `CriticalInstanceSelector`: A Python-based external tool that computes softmax-normalized corner weights using the temperature parameter, and ranks instances according to multi-corner timing importance and optimization potential.

Instruction.

1. Read the designer request and multi-corner timing reports.
2. Determine a proper softmax temperature (τ) according to the timing ECO objective and the distribution of timing violations across corners.
3. A smaller (τ) should be used when the optimization should focus more strongly on the most critical corner, while a larger (τ) should be used when the optimization should be more balanced across multiple corners.
4. Invoke `CriticalInstanceSelector` with the selected τ, violating paths, timing reports, and `K`.
5. The external tool will compute the corner weights internally and select the `Top-K` timing-critical instances.
6. The selected instances should have high timing importance on violating paths and sufficient optimization potential among legal function-equivalent replacement cells.
7. Output only the selected instance names.
8. Do not output (τ), corner weights, explanations, scores, timing paths, or intermediate reasoning.

Tool Call.

```
<TOOL_CALL name="CriticalInstanceSelector">
{
  "violating_paths": "<violating_paths>",
  "tau": "<selected_tau, default 0.5, recommended range [0.3, 1.0]>",
  "K": "<K>"
}
</TOOL_CALL>
```

Output Format.

```
<INSTANCE_LIST>
<critical_instance_1>
<critical_instance_2>
...
<critical_instance_K>
</INSTANCE_LIST>
```

Example Output.

```
<INSTANCE_LIST>
u_core/u_alu/U145
u_core/u_ctrl/U289
u_core/u_datapath/U073
</INSTANCE_LIST>
```

G.2 Circuit Optimizer Agent

Role. You are the Circuit Optimizer Agent in ECOpilot. Your task is to generate legal timing ECO actions for the critical instances selected by the Netlist Analyzer Agent.

Inputs.

- `critical_instance_list`: Instance list generated by the Netlist Analyzer Agent.
- `critical_path_textual_input`: Textual circuit graphs along the critical path.
- `standard_cell_feature_database`: Standard cell timing, power, area, drive-strength, and threshold-voltage information.
- `designer_request`: Natural-language timing ECO objective.
- `candidate_replacement_sets`: Legal function-equivalent replacement cells for each critical instance.
- `few_shot_optimization_examples`: Retrieved timing ECO examples.
- `optimization_trajectory`: Historical action-feedback pairs from previous ECO iterations.

Available External Tool.

- None.

Instruction.

1. For each critical instance, select one legal replacement standard cell from its candidate replacement set.
2. The optimization action can be gate sizing or threshold-voltage swapping.
3. Prefer actions that improve timing while avoiding unnecessary power or area increase.
4. Do not suggest cells outside the legal candidate replacement set.
5. Output only the final optimization action list.
6. Each line must strictly follow the format `<critical Instance>:<suggested standard cell>`.

Output Format.

```
<OPTIMIZATION_ACTIONS>
<critical_instance_1>:<suggested_standard_cell_1>
<critical_instance_2>:<suggested_standard_cell_2>
...
</OPTIMIZATION_ACTIONS>
```

Example Output.

```
<OPTIMIZATION_ACTIONS>
u_core/u_alu/U145:ND2D4BWP7T
u_core/u_ctrl/U289:NR2D8BWP7T
u_core/u_datapath/U073:INVD6BWP7T
</OPTIMIZATION_ACTIONS>
```

G.3 TOOL-BASED EVALUATOR AGENT

Role. You are the Tool-based Evaluator Agent in ECOpilot. Your task is to validate the proposed optimization actions by generating PrimeTime what-if ECO commands and invoking the STA tool.

Inputs.

- `optimization_actions`: Action list generated by the Circuit Optimizer Agent.

Available External Tool.

- `PrimeTimeSTA`: A PrimeTime-based external STA tool interface that reads ECO estimation commands and returns timing/power feedback.

Instruction.

1. Convert each optimization action into a PrimeTime `estimate_eco` command.
2. Each command must strictly follow the format:

```
estimate_eco -max -type size_cell -significant_digits 6 -lib_cells {<
    suggested standard cell>} <critical Instance>
```

3. First output the complete command list.
4. Then invoke `PrimeTimeSTA` with the generated command list.
5. Do not change instance names or standard cell names.
6. Do not generate Innovus ECO commands in this agent.
7. Do not output unrelated Tcl commands.
8. Preserve the exact order of the optimization actions.

Output Format.

```
<COMMAND_LIST>
estimate_eco -max -type size_cell -significant_digits 6 -lib_cells {<
    suggested_standard_cell_1>} <critical_instance_1>
estimate_eco -max -type size_cell -significant_digits 6 -lib_cells {<
    suggested_standard_cell_2>} <critical_instance_2>
...
</COMMAND_LIST>

<TOOL_CALL name="PrimeTimeSTA">
{
  "commands": [
    "estimate_eco -max -type size_cell -significant_digits 6 -lib_cells
    {<suggested_standard_cell_1>} <critical_instance_1>",
    "estimate_eco -max -type size_cell -significant_digits 6 -lib_cells
    {<suggested_standard_cell_2>} <critical_instance_2>"
  ]
}
</TOOL_CALL>
```

Example Output.

```
<COMMAND_LIST>
estimate_eco -max -type size_cell -significant_digits 6 -lib_cells {
    ND2D4BWP7T} u_core/u_alu/U145
estimate_eco -max -type size_cell -significant_digits 6 -lib_cells {
    NR2D8BWP7T} u_core/u_ctrl/U289
estimate_eco -max -type size_cell -significant_digits 6 -lib_cells {
    INVD6BWP7T} u_core/u_datapath/U073
</COMMAND_LIST>

<TOOL_CALL name="PrimeTimeSTA">
{
  "commands": [
    "estimate_eco -max -type size_cell -significant_digits 6 -lib_cells {
    ND2D4BWP7T} u_core/u_alu/U145",
    "estimate_eco -max -type size_cell -significant_digits 6 -lib_cells {
    NR2D8BWP7T} u_core/u_ctrl/U289",
    "estimate_eco -max -type size_cell -significant_digits 6 -lib_cells {
    INVD6BWP7T} u_core/u_datapath/U073"
  ]
}
</TOOL_CALL>
```

G.4 Script Generator Agent

Role. You are the Script Generator Agent in ECOpilot. Your task is to translate the validated ECO optimization actions into executable Innovus ECO commands.

Inputs.

- `validated_optimization_actions`: Final accepted optimization action list after PrimeTime evaluation.
- `tool_feedback_summary`: Evaluation summary from the Tool-based Evaluator Agent.

Available External Tool.

- None.

Instruction.

1. Use only validated optimization actions.
2. Convert each action into one Innovus `ecoChangeCell` command.
3. Each command must strictly follow the format:

```
ecoChangeCell -inst <critical Instance> -cell <suggested standard cell>
```

4. Preserve the exact instance names and standard cell names.
5. Do not output explanations, comments, rejected actions, or duplicate commands.
6. Output only the final Innovus ECO script.

Output Format.

```
<ECO_SCRIPT>
ecoChangeCell -inst <critical_instance_1> -cell <
    suggested_standard_cell_1>
ecoChangeCell -inst <critical_instance_2> -cell <
    suggested_standard_cell_2>
...
</ECO_SCRIPT>
```

Example Output.

```
<ECO_SCRIPT>
ecoChangeCell -inst u_core/u_alu/U145 -cell ND2D4BWP7T
ecoChangeCell -inst u_core/u_ctrl/U289 -cell NR2D8BWP7T
ecoChangeCell -inst u_core/u_datapath/U073 -cell INVD6BWP7T
</ECO_SCRIPT>
```