HIERRTL: HIERARCHICAL GRAPH REPRESENTATION LEARNING FOR MULTI-GRANULARITY RTL PERFORMANCE MODELING

Anonymous authors
Paper under double-blind review

ABSTRACT

Accurate power, performance, and area (PPA) assessment requires time-consuming synthesis and physical-design flows, limiting design-space exploration at the register-transfer level (RTL). Fast RTL prediction must capture both fine-grained hardware structure and design-wide context while preserving the association between logic and the state updates it supports. We propose HierRTL, a hierarchical Graph U-Net for joint design-level PPA and register-level timing prediction. HierRTL organizes RTL graphs into overlapping register cones that capture the data and control dependencies of individual next-state updates, with a directed cone graph modeling dependencies between registers. A U-shaped encoder–decoder aggregates information from nodes to cones and the design, then routes context back to nodes through the same cone memberships. This bidirectional hierarchy preserves shared-logic associations and produces contextualized node representations for predicting design-level area, power, worst negative slack, and total negative slack, together with register-level setup slack. To support local RTL optimization, we construct bounded local rewrites, verify them against the original RTL, and evaluate accepted implementations with a common flow. These equivalence-verified results define a soft opportunity target for each editable region. Starting from the jointly supervised predictor, HierRTL encodes the unmodified design once, summarizes each region only at readout, and learns four listwise rankings over the regions of the same design. Candidate implementations define supervision but are not required as model inputs. This connects multi-granularity RTL performance modeling with the practical task of prioritizing regions that are more likely to yield high optimization value before expensive search.

1 INTRODUCTION

Power, performance, and area (PPA) are primary quality metrics in digital integrated-circuit design. Decisions made at the register-transfer level (RTL) substantially influence implementation quality, yet accurate PPA assessment typically requires time-consuming synthesis and physical-design flows. Repeatedly invoking these flows during RTL development creates a long feedback loop and limits design-space exploration. Fast pre-synthesis PPA prediction can provide earlier feedback, helping designers assess alternative implementations and direct optimization effort more efficiently (Fang et al., 2023; 2024). In a design with many editable regions, this also raises a more actionable question: when rewrite, synthesis, and verification budgets are limited, which regions should be explored first?

Recent studies use large language models (LLMs) and pre-trained code models to predict implementation metrics directly from Verilog code (Liu et al., 2025a;c). These approaches benefit from representations learned from code, but RTL describes hardware whose operators, registers, and connectivity jointly determine implementation quality. Token sequences and module-level embeddings do not explicitly expose these structural relationships. Moreover, PPA-related information spans multiple granularities: primitive operations characterize local logic, register-to-register dependencies organize state updates, and design-wide interactions provide the context in which local structures are implemented. Effective RTL modeling therefore calls for representations that connect these complementary scales.

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.

Graph-based methods make RTL structure explicit through representations such as abstract-syntax, operator, and bit-level graphs (Fang et al., 2023; Liu et al., 2025b). Hierarchical GNNs, register-centered subcircuits, and multimodal circuit representations further support modeling beyond individual nodes (Xu et al., 2023; Fang et al., 2025b;a; Zhao et al., 2026). A central challenge, however, is to integrate design-wide context into fine-grained features while retaining the association between logic and the state updates it supports. A flat graph may require many local message-passing steps to connect distant logic (Alon & Yahav, 2020), whereas direct pooling into a design vector collapses node-specific associations. Hierarchical aggregation provides an intermediate scale, but aggregation alone does not specify how the resulting context should return to the relevant implementation nodes. This motivates a hierarchy that supports both upward abstraction and structured downward contextualization.

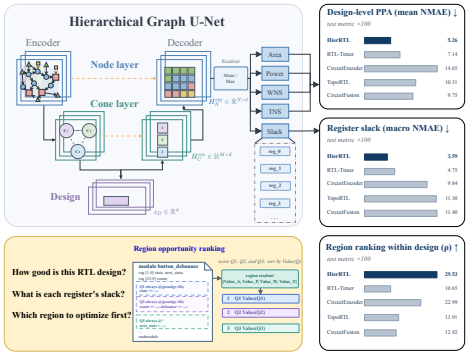


Figure 1: Three outputs of HierRTL from one RTL hierarchy: design-level area/power/WNS/TNS, per-register setup slack, and within-design ranking of optimization regions. The bars compare test PPA mean NMAE, register macro NMAE, and region-ranking Spearman ρ, respectively (all multiplied by 100). Equivalent rewrites supply ranking labels offline but are not inference inputs.

To address this challenge, we propose **HierRTL**, a hierarchical Graph U-Net for multi-granularity RTL PPA and timing modeling. HierRTL organizes an RTL graph around *register cones*, which capture the data and control dependencies supporting individual next-state updates. Cones retain overlapping memberships so that shared logic can support multiple state updates, and a directed cone graph represents dependencies between registers. A U-shaped encoder–decoder exchanges information along the path node $\rightarrow$ cone $\rightarrow$ design $\rightarrow$ cone $\rightarrow$ node. The encoder aggregates fine-grained logic into register-cone and design representations, while the decoder routes context back through the same node–cone memberships. Learned membership weights and gated skip fusion combine this context with fine-grained features. Design-level area, power, worst negative slack (WNS), and total negative slack (TNS) are read out from the decoded nodes, while a register-level head predicts the worst setup slack of each labeled register. Joint supervision thus trains a shared representation at both design and register granularities.

Beyond estimating implementation metrics, local RTL optimization must decide where limited optimization effort should be spent. We use bounded local rewrites to measure this otherwise latent opportunity: candidates are generated inside a selected region, checked against the surrounding source and interface contract, verified for functional equivalence, and evaluated by a common implementation flow. Their measured gains define a soft-oracle opportunity target for each region and each PPA objective. Starting from the jointly trained PPA and timing model, HierRTL encodes the unmodified design and applies region membership only at the final readout. Four independent heads are then fine-tuned with a listwise objective to rank the regions of the same design. This formulation learns which regions deserve early search budget without injecting baseline PPA, candidate identity, or candidate graphs into message passing.

The distinction is intentional: the downstream model answers *where to search* rather than *how to rewrite*. A high opportunity score is therefore a prioritization signal, not a guarantee that a particular unseen rewrite will succeed or attain a fixed gain. It is useful precisely when exhaustive rewrite generation, equivalence checking, and implementation evaluation are too expensive to apply to every region.

We evaluate HierRTL on a 22,848-design RTL corpus and on 21,914 equivalence-verified local rewrites, assessing design-level PPA prediction, register-level timing prediction, and design-internal region prioritization. Our contributions are summarized as follows:

- We introduce HierRTL, a register-centered hierarchical Graph U-Net that performs bidirectional representation learning across node, register-cone, and design scales for joint design-level PPA and register-level timing prediction.

- To the best of our knowledge, we present the first learning-based formulation of RTL region-level optimization opportunity ranking, using equivalence-verified rewrites as supervision while requiring only the original RTL graph at inference.
- HierRTL establishes strong performance across all three granularities: it reduces mean PPA NMAE by 26.3% over the strongest baseline, achieves the lowest register slack NMAE, and consistently outperforms existing methods on within-design region ranking.

2 Related Works

2.1 RTL Learning

LLM-based. Recent studies apply large language models and pre-trained code models to learn RTL representations directly from hardware description languages. CodeV constructs instruction-tuning data by generating multi-level natural-language summaries from high-quality Verilog code, improving the capability of LLMs for RTL generation (Zhao et al., 2024). DeepRTL further aligns Verilog code with line-, block-, and module-level descriptions and adopts curriculum learning to unify RTL understanding and generation (Liu et al., 2025a). DeepRTL2 extends this paradigm by jointly supporting generation- and embedding-based tasks, including code retrieval, functional equivalence checking, and performance prediction (Liu et al., 2025c). Despite their strong semantic modeling capabilities, these approaches primarily represent RTL as token sequences and therefore lack explicit inductive biases for hardware topology and register dependencies.

Graph-based. Graph-based approaches explicitly represent the structural properties of RTL designs. MasterRTL converts RTL into a bit-level simple operator graph and develops specialized models for pre-synthesis timing, power, and area estimation (Fang et al., 2023). RTL-Timer targets fine-grained timing at register endpoints (Fang et al., 2024). CircuitEncoder learns a cross-stage-aligned circuit representation (Fang et al., 2025b), whereas StructRTL pretrains on control-data-flow graphs and can distill information from mapped netlists (Liu et al., 2025b). TopoRTL decomposes a design into register cones and introduces topology-aware positional encodings to capture structural differences among subcircuits and complete designs (Zhao et al., 2026). These methods demonstrate the importance of circuit topology for RTL performance modeling. However, existing graph representations either operate on a flat fine-grained graph or mainly aggregate local structures into a global representation, without explicitly decoding design-level context back to the underlying implementation nodes.

Multimodal. Multimodal RTL learning combines complementary behavioral and structural representations. CircuitFusion jointly encodes RTL code, structural graphs, and natural-language functionality summaries, and employs contrastive alignment within and across modalities and design stages (Fang et al., 2025a). UniRTL unifies RTL code and control-data-flow graphs through mutual masked modeling and staged multimodal alignment (Liu et al., 2026). By integrating code semantics with graph topology, these methods improve representation generality across downstream tasks. Nevertheless, their modality alignment is primarily designed to learn unified subcircuit- or design-level embeddings, rather than to support bidirectional information propagation and prediction across multiple structural granularities.

2.2 Hierarchical Learning

Hierarchical graph learning aims to capture structures at different resolutions by progressively pooling fine-grained nodes into coarser representations. Generic approaches such as differentiable graph pooling and Graph U-Nets learn graph coarsening operations from data and use the resulting representations for graph- or node-level tasks (Ying et al., 2018; Gao & Ji, 2019). In RTL learning, SNS v2 introduces a two-level hierarchical GNN that first encodes register-centered subgraphs and then models dependencies among registers for design-level prediction (Xu et al., 2023). Register-cone representations are also adopted by CircuitFusion and TopoRTL to bridge local state-update logic and complete circuits (Fang et al., 2025a; Zhao et al., 2026). However, learned pooling may not preserve the semantic ownership of RTL logic, while existing RTL-specific hierarchies predominantly follow a bottom-up aggregation process. In contrast, HierRTL constructs a deterministic node–cone–design hierarchy from register-centered dataflow and introduces a U-shaped encoder–decoder with skip connections. This architecture not only abstracts local logic into a compact

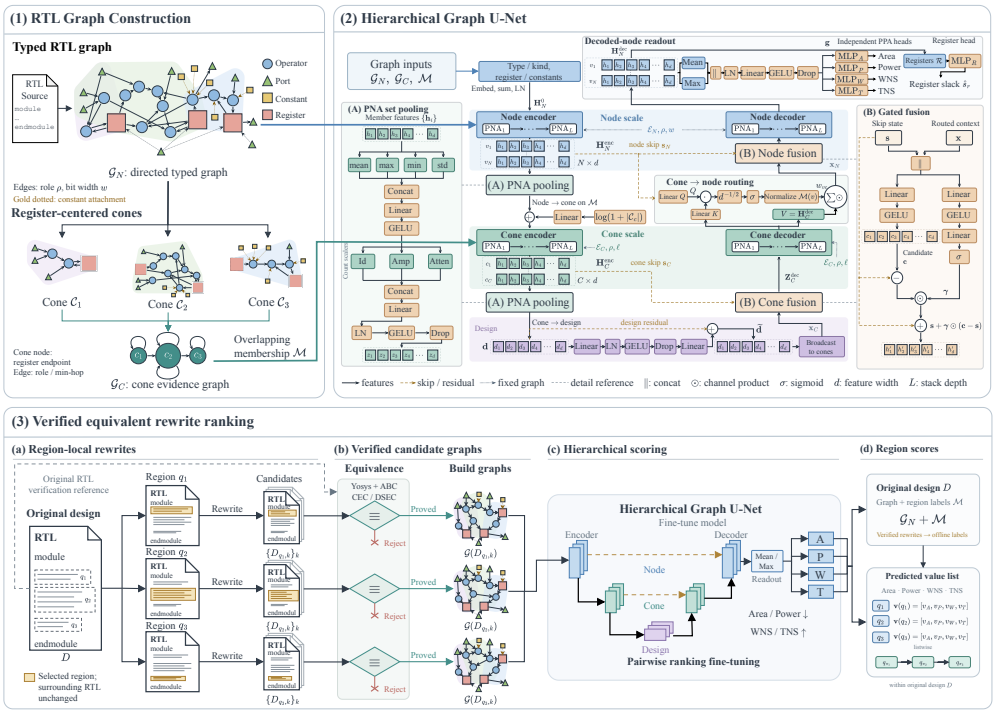


Figure 2: Overview of HierRTL . (1) RTL becomes a typed implementation graph with overlapping register-centered cones and a sparse cone-dependency graph. (2) A PNA-based Graph U-Net passes context from nodes to cones to the design and back, predicting area, power, WNS, TNS, and per-register setup slack. (3) Equivalent local rewrites are verified and evaluated offline to derive four region-opportunity targets. Inference ranks regions using only the original graph and region membership at readout, not candidate graphs.

register-interaction graph but also routes design context back to the corresponding cones and implementation nodes, enabling joint design-, register-, and subgraph-level performance modeling.

3 Method

3.1 Overview

Given an RTL design, HierRTL constructs a typed implementation graph and a sparse graph of register-centered logic cones. Its hierarchical encoder first propagates information from implementation nodes to register updates and then to the design level. A symmetric decoder routes global context back to the nodes through the same semantic hierarchy. The resulting node representations are jointly supervised by design-level PPA and register-level setup slack.

We further adapt this representation to prioritize local RTL optimization. Equivalent rewrites are generated and evaluated offline to estimate the optimization opportunity of each region. At inference, the model encodes only the original design and scores each queried region at readout, without generating or evaluating rewrite candidates.

To make the construction concrete, Fig. 3 follows an example RTL fragment from its semantic endpoints to the typed implementation graph and the overlapping register-centered cones used by the model.

3.2 Hierarchical Graph U-Net

The encoder–decoder follows the graph U-Net principle of coupling coarse representations with a return path to fine-grained nodes (Gao & Ji, 2019); here the levels and memberships are fixed by RTL register dependencies rather than learned graph partitions.

Register-Centered RTL Hierarchy Let $\mathcal{G}_N = (\mathcal{V}, \mathcal{E})$ be a directed, typed RTL graph whose nodes represent implementation primitives, ports, constants, and registers. For each target register r, we trace data and control dependencies backward from its data input and stop at upstream registers. The resulting support defines a register cone

$$\mathcal{C}_r = \{v \in \mathcal{V} \mid v \rightsquigarrow r_D, \text{ with no internal register on the path}\}. \tag{1}$$

The cone represents the RTL logic supporting one next-state update rather than a gate-level timing path. Because shared logic may contribute to several updates, cones are allowed to overlap. We preserve this overlap with a fixed node–cone membership relation $\mathcal{M} \subseteq \mathcal{V} \times \mathcal{C}$.

We connect two cones when an upstream register associated with one cone provides boundary evidence for the other. These directed dependencies form a sparse cone graph $\mathcal{G}_C = (\mathcal{C}, \mathcal{E}_C)$. The resulting hierarchy explicitly links implementation logic, state updates, and the whole design, enabling long-range communication without discarding the logic support of individual registers.

Bidirectional Hierarchical Propagation The model follows a U-shaped information path

$$\text{node} \rightarrow \text{cone} \rightarrow \text{design} \rightarrow \text{cone} \rightarrow \text{node}. \tag{2}$$

On the upward path, a graph encoder first updates node states on $\mathcal{G}_N$. Node states are then aggregated over $\mathcal{M}$ to initialize their associated cones, followed by propagation on $\mathcal{G}_C$ and aggregation into a design state:

$$\begin{aligned} \mathbf{H}_N^{\downarrow} &= \Phi_N(\mathbf{H}_N^0, \mathcal{E}), \\ \mathbf{z}_c &= P_{N\to C}(\{\mathbf{h}_v^{\downarrow} : (v,c) \in \mathcal{M}\}), \\ \mathbf{H}_C^{\downarrow} &= \Phi_C(\mathbf{Z}_C, \mathcal{E}_C), \\ \mathbf{d} &= P_{C\to D}(\{\mathbf{h}_c^{\downarrow} : c \in \mathcal{C}\}). \end{aligned} \tag{3}$$

Here Φ_N and Φ_C denote relation-aware graph propagation, while $P_{N\to C}$ and $P_{C\to D}$ are permutation-invariant aggregators. Their specific implementations are not essential to the hierarchy. Our main instantiation uses Principal Neighbourhood Aggregation at the node and cone scales (Corso et al., 2020).

On the downward path, the design state is fused with each cone's skip state and propagated over the cone graph. Context is then returned to nodes through their fixed memberships. Since a node may belong to multiple cones, we learn a membership weight for each node–cone pair. The scaled dot-product score uses the standard attention parameterization (Vaswani et al., 2017):

$$\begin{aligned} a_{vc} &= \langle Q\mathbf{h}_v^{\downarrow}, K\mathbf{h}_c^{\uparrow} \rangle / \sqrt{d}, \\ w_{vc} &= \frac{\sigma(a_{vc})}{\sum_{c':(v,c')\in\mathcal{M}} \sigma(a_{vc'})}, \qquad \mathbf{q}_v = \sum_{c:(v,c)\in\mathcal{M}} w_{vc}\mathbf{h}_c^{\uparrow}. \end{aligned} \tag{4}$$

The resulting context is gated with the node skip state before the final node update. This decoder contextualizes implementation nodes with both their associated state updates and the entire design, while never introducing node–cone links absent from the RTL structure.

Multi-Granularity Supervision The decoded node states support two complementary prediction tasks. A design-level readout feeds four independent heads for area, power, worst negative slack (WNS), and total negative slack (TNS). A node-level head predicts the worst setup slack of each labeled register. The joint objective is

$$\begin{aligned} \mathcal{L}_{\text{reg}} &= \frac{1}{|\mathcal{D}^+|} \sum_{i\in\mathcal{D}^+} \frac{1}{|\mathcal{R}_i^+|} \sum_{r\in\mathcal{R}_i^+} (\widehat{s}_r - s_r)^2, \\ \mathcal{L} &= \mathcal{L}_{\text{PPA}} + \lambda \mathcal{L}_{\text{reg}}. \end{aligned} \tag{5}$$

The per-design normalization in $\mathcal{L}_{\text{reg}}$ prevents designs with many labeled registers from dominating training. Together, the two tasks supervise the same representation at design and register scales.

3.3 Region-Level Opportunity Ranking from Verified Rewrites

Local optimization changes only a bounded part of an RTL design. Rather than predicting an implementation in isolation, we rank regions by their expected value for subsequent rewrite, synthesis, and verification. The model is a prioritization mechanism: it neither generates rewrites nor guarantees the gain of an unseen candidate.

Verified-Rewrite Supervision For an original design D and an editable region q, we generate a bounded set of local rewrites

$$\mathcal{X}_{D,q} = \{D, D_{q,1}, \ldots, D_{q,K_q}\}, \tag{6}$$

where D is the shared no-rewrite baseline. Edits are restricted to q, and the surrounding source and interface must remain unchanged. Each candidate is processed by the same synthesis and equivalence-checking flow. Only rewrites explicitly proved equivalent are retained; failed, inconclusive, and timed-out checks are excluded. A common implementation flow then measures area, power, WNS, and TNS. Data splits are grouped by original design so that rewrites of the same design cannot cross training, validation, and test sets.

This procedure produces a bounded collection of equivalence-verified candidates offline; the concrete candidate-construction, verification, and feedback procedure is described in Appendix A.1. The trained predictor does not require candidate generation or a language model at inference.

For metric m, let $g_{D,q,k,m} \geq 0$ be the measured improvement of candidate k over the baseline. We summarize the candidates with a soft-oracle target

$$\begin{aligned} \alpha_{D,q,k,m} &= \frac{\exp(g_{D,q,k,m}/T_m)}{\sum_{j\in\{0\}\cup\mathcal{K}_{D,q}} \exp(g_{D,q,j,m}/T_m)}, \\ z_{D,q,m} &= \sum_{k\in\{0\}\cup\mathcal{K}_{D,q}} \alpha_{D,q,k,m} g_{D,q,k,m}, \qquad g_{D,q,0,m} = 0. \end{aligned} \tag{7}$$

This target captures the overall opportunity demonstrated by verified candidates while reducing sensitivity to a single extreme rewrite. In our implementation, $T_m = 0.1 s_m$, where s_m is the 90th percentile of positive candidate gains for target m on the training split; the zero-gain baseline is included in the softmax.

Baseline-Graph Region Ranking At inference, the hierarchical encoder processes the unmodified design once. A region q is specified only by its node set $\mathcal{V}_{D,q}$ at readout. Its local summary is combined with a global design summary and passed to four independent metric-specific heads:

$$\begin{aligned} \boldsymbol{\ell}_{D,q} &= P_R(\{\mathbf{h}_v^{\uparrow} : v \in \mathcal{V}_{D,q}\}), \\ \widehat{z}_{D,q,m} &= f_m([\boldsymbol{\ell}_{D,q} \| \mathbf{g}_D]), \qquad m \in \{1,2,3,4\}. \end{aligned} \tag{8}$$

Region identifiers, candidate graphs, baseline PPA values, and measured gains are not injected into graph message passing. Candidate implementations are used only to construct supervision.

Within each original design, the soft-oracle targets are converted to rank relevance values $r_{D,q,m}$. For each metric, the target and predicted region distributions are

$$\pi^*_{D,q,m} = \mathrm{softmax}_{q\in\mathcal{Q}(D)}(r_{D,q,m}/T), \qquad \pi_{D,q,m} = \mathrm{softmax}_{q\in\mathcal{Q}(D)}(\widehat{z}_{D,q,m}/T). \tag{9}$$

The four heads are trained with listwise cross-entropy, averaged first within each design and then across designs, following the listwise learning-to-rank formulation (Cao et al., 2007). At inference, their scores rank regions within the same design. They represent relative optimization opportunity, not calibrated predictions of the PPA gain from an unseen rewrite.

4 Experiments

4.1 Experimental Setup

Datasets and splits. We evaluate two linked tasks using splits at the original-design level. The complete accounting is provided in Appendix Table 4, and the source-wise size distribution is summarized in Appendix Table 5. Stage I contains 22,848 register-bearing RTL designs, divided into 18,270/2,272/2,306 training/validation/test designs, with 151,305 valid register-slack labels. Stage II contains 1,919 designs (1,547/170/202 by split), 6,093 target regions, and 21,914 equivalence-verified rewrites. No original design or its rewrites crosses a split. The tables report counts after deterministic filtering and the split is performed before any candidate-derived target is used.

Data processing and labels. At a high level, RTL is normalized into a typed implementation graph and register-centered cones for HierRTL ; Area, Power, WNS, and TNS are obtained from the

Table 1: Stage-I test PPA and register-slack prediction by method family. Avg. is the mean across the four PPA targets.

Method	N	Area		Power		WNS		TNS		Avg.		Slack	
		NMAE ↓	ρ ↑	NMAE ↓	ρ ↑	NMAE ↓	ρ ↑	NMAE ↓	ρ ↑	NMAE ↓	ρ ↑	NMAE ↓	ρ ↑
LLM-based methods													
Qwen3-E-0.6B	22848	12.72	91.04	16.38	89.21	10.73	86.45	15.61	89.66	13.86	89.09	–	–
Qwen3-E-4B	22848	11.39	93.10	14.37	90.67	9.92	89.18	16.01	91.58	12.92	91.13	–	–
Qwen3-E-8B	22848	10.87	93.32	15.99	89.84	11.64	86.14	15.62	91.51	13.53	90.20	–	–
CodeV-CL-7B	22848	11.82	93.46	15.38	91.78	10.60	87.93	14.18	92.47	13.00	91.41	–	–
CodeV-DS-6.7B	22848	12.10	93.38	16.57	90.43	11.51	88.06	14.77	92.69	13.74	91.14	–	–
CodeV-QC-7B	22848	10.69	94.39	13.71	92.70	9.81	89.76	14.86	92.83	12.27	92.42	–	–
GNN-based methods													
GCN	22848	11.01	92.58	14.37	91.96	8.12	92.71	15.10	92.56	12.15	92.45	12.01	42.92
GAT	22848	10.30	93.00	13.32	92.22	7.59	92.97	14.50	93.03	11.43	92.81	12.02	43.78
GIN	22848	14.21	88.65	16.48	88.18	10.15	90.41	16.49	89.38	14.33	89.15	12.16	43.09
EDA-based methods													
RTL-Timer	22848	5.71	97.25	7.02	96.94	8.18	93.17	7.66	96.34	7.14	95.93	4.75	**68.60**
CircuitEncoder	22798	14.20	90.44	17.43	90.35	10.34	89.13	16.64	89.22	14.65	89.79	9.84	58.07
StructRTL	22829	19.91	89.73	19.70	88.45	21.03	84.53	20.87	88.11	20.38	87.71	15.93	43.46
TopoRTL	22848	8.68	95.67	10.80	94.28	10.64	89.02	11.13	94.21	10.31	93.30	11.30	48.92
UniRTL	22829	8.65	97.17	11.60	96.09	10.21	91.69	9.71	96.22	10.04	95.29	–	–
CircuitFusion	19758	8.04	97.07	11.04	95.83	10.65	90.94	9.28	96.05	9.75	94.97	11.40	50.96
Ours													
HierRTL	22848	**4.32**	**98.28**	**6.21**	**97.95**	**5.07**	**96.50**	**5.43**	**97.66**	**5.26**	**97.60**	**3.59**	66.21

common implementation flow, and aligned register slack is masked when no valid endpoint label is available. For Stage II, bounded local RTL replacements are syntactically checked, formally verified for equivalence, and evaluated against the unchanged baseline to define soft opportunity targets. Candidates construct supervision only; inference uses the original graph and a region readout. Full processing, verification, and target construction details are provided in Appendix A.1.

Compared methods and evaluation. The comparison includes LLM-based, generic GNN-based, and EDA-based methods. The LLM baselines include three Qwen3-Embedding models and three variants from the published CodeV family (Zhao et al., 2024), the generic controls instantiate GCN, GAT, and GIN (Kipf & Welling, 2016; Veličković et al., 2017; Xu et al., 2018), and the EDA-oriented comparisons include RTL-Timer, CircuitEncoder, StructRTL, TopoRTL, UniRTL, and CircuitFusion (Fang et al., 2024; 2025b; Liu et al., 2025b; Zhao et al., 2026; Liu et al., 2026; Fang et al., 2025a). The Stage-I reference is the dual-PNA Hierarchical Graph U-Net with upward PNA propagation, sigmoid pooling, gated downward fusion, four independent PPA heads, and a register-slack head. We use AdamW and validation prefix-best checkpoints within a 400-epoch budget over two random seeds. Stage II fixes the soft-oracle, listwise objective, and one-layer region-attention readout and uses three random seeds. We report target-wise PPA NMAE and Spearman correlation, macro register-slack error/correlation, and within-design region ranking metrics. Values in the result tables are multiplied by 100 for readability; definitions and hyperparameters are summarized in Appendix Section A.2.

4.2 Stage I: Multi-Level PPA Prediction

Design-level PPA prediction. HierRTL consistently outperforms the competing methods across all four PPA targets. It achieves an average NMAE of 5.26, representing a 26.3% relative reduction over RTL-Timer, the strongest baseline with an average NMAE of 7.14. It also obtains the highest average Spearman correlation of 97.60. Moreover, HierRTL achieves the lowest NMAE and the highest Spearman correlation for Area, Power, WNS, and TNS, demonstrating consistent improvements in both prediction accuracy and rank preservation across physical and timing metrics.

Register-level timing prediction. Register slack is supervised only for RTL registers that can be reliably aligned with valid implementation timing paths, while unmatched endpoints are excluded through masking. As shown in Table 1, HierRTL achieves the lowest register-level NMAE of 3.59, reducing the error by 24.4% relative to RTL-Timer, which obtains the second-lowest NMAE of 4.75.

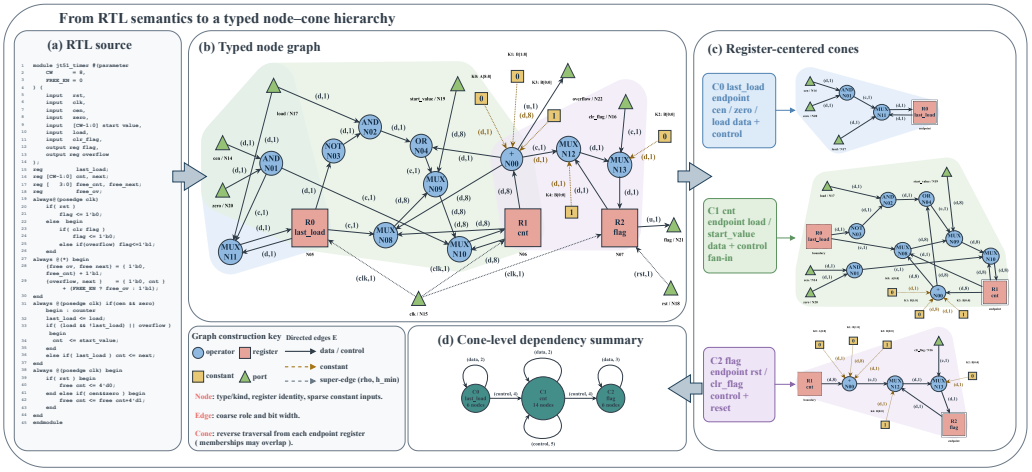


Figure 3: Example RTL-to-hierarchy construction. (a) RTL source provides state-update endpoints. (b) Operators, registers, ports, and constants form typed nodes; directed edges encode data/control role and bit width, while dotted links attach constants. (c) Reverse traversal from each endpoint builds its support cone; shared logic can belong to multiple cones. (d) Directed cone edges summarize register-to-register dependencies by role and minimum hop count. Cone membership follows dependencies, not source-line ranges.

In Spearman correlation, HierRTL is second only to RTL-Timer and remains relatively close (66.21 versus 68.60). Together with its lower slack NMAE, this makes HierRTL the strongest overall model for register-level timing prediction.

4.3 Stage II: Region-Level Ranking

Overall ranking performance. HierRTL achieves the strongest performance across all four aggregate ranking metrics. It obtains a Spearman correlation of 29.53 and a Kendall correlation of 26.93, exceeding the strongest baseline results of 26.03 and 23.55, respectively. It also improves Top-1 accuracy from the strongest baseline value of 45.61 to 46.88 and reduces normalized regret from 49.15 to 46.37. These results show that HierRTL improves both the global ordering of regions and the quality of the highest-ranked selection, which is particularly important when only a limited number of regions can be evaluated through expensive rewrite and implementation flows.

Target-wise ranking performance. The improvements are consistent across all four optimization targets. HierRTL achieves Spearman correlations of 30.65, 33.06, 25.69, and 28.72 for Area, Power, WNS, and TNS, respectively, outperforming all baselines for every target. This consistency indicates that the learned representation captures region-level optimization opportunities associated with both physical cost and timing quality rather than specializing in a single PPA objective.

4.4 Ablation Studies

Hierarchy construction. Table 3 summarizes the ablations. Removing the cone level raises mean PPA NMAE from 5.26 to 5.67 and lowers Stage-II Spearman from 29.53 to 25.90; removing the design level as well changes them to 6.81 and 20.63. These results show that both cone-level aggregation and design-level context are important for supporting global PPA prediction, local timing estimation, and region ranking.

Semantic cone representation. Fixed-hop and METIS (Karypis & Kumar, 1998) groupings trail semantic cones (PPA NMAE 5.58/5.95; ranking Spearman 21.40/22.23). Thus, topology-only hierarchies provide some benefit over a flat representation but cannot replace cones constructed from register-centered data and control dependencies; semantic cones are important for transferring localized circuit information to region-level optimization decisions.

Inter-level propagation and fusion. Removing upward PNA slightly improves mean PPA NMAE (5.19 versus 5.26) but worsens register NMAE (3.73 versus 3.59) and sharply reduces ranking Spearman (19.42 versus 29.53). Removing decoder residual fusion raises mean PPA NMAE to 5.63 and

Table 2: Stage-II test ranking of regions within each original design. we report results with mean ± standard deviation.

Method	Overall Ranking				Target-wise Spearman ρ			
	$\rho \uparrow$	$\tau \uparrow$	Top-1 ↑	Norm. Regret ↓	Area ↑	Power ↑	WNS ↑	TNS ↑
GNN-based methods								
GCN	$22.97_{\pm 2.32}$	$20.76_{\pm 1.87}$	$39.98_{\pm 2.27}$	$52.95_{\pm 2.19}$	$22.48_{\pm 4.75}$	$20.59_{\pm 2.63}$	$25.36_{\pm 5.77}$	$23.44_{\pm 7.90}$
GAT	$26.03_{\pm 5.71}$	$23.55_{\pm 5.12}$	$42.37_{\pm 2.49}$	$50.37_{\pm 1.75}$	$29.45_{\pm 3.11}$	$27.46_{\pm 5.83}$	$21.16_{\pm 10.37}$	$26.05_{\pm 9.76}$
GIN	$24.15_{\pm 2.16}$	$21.84_{\pm 1.78}$	$43.31_{\pm 0.49}$	$49.15_{\pm 0.83}$	$21.54_{\pm 2.43}$	$30.61_{\pm 1.45}$	$23.68_{\pm 7.36}$	$20.78_{\pm 4.06}$
EDA-based methods								
RTL-Timer	$10.65_{\pm 3.87}$	$10.00_{\pm 3.42}$	$36.80_{\pm 0.90}$	$56.04_{\pm 1.73}$	$5.47_{\pm 6.81}$	$11.80_{\pm 2.60}$	$14.63_{\pm 7.29}$	$10.71_{\pm 7.38}$
CircuitEncoder	$22.99_{\pm 2.55}$	$21.06_{\pm 2.32}$	$40.06_{\pm 2.33}$	$51.78_{\pm 1.69}$	$25.53_{\pm 2.83}$	$28.70_{\pm 7.62}$	$15.84_{\pm 2.03}$	$21.89_{\pm 5.20}$
StructRTL	$6.49_{\pm 8.29}$	$6.27_{\pm 7.76}$	$36.43_{\pm 1.64}$	$57.13_{\pm 1.69}$	$8.89_{\pm 7.91}$	$6.96_{\pm 19.71}$	$6.48_{\pm 6.08}$	$3.62_{\pm 3.44}$
TopoRTL	$13.91_{\pm 2.94}$	$13.24_{\pm 2.59}$	$40.86_{\pm 1.57}$	$53.07_{\pm 1.45}$	$13.28_{\pm 5.81}$	$9.91_{\pm 2.07}$	$17.32_{\pm 2.96}$	$15.13_{\pm 4.06}$
CircuitFusion	$12.92_{\pm 2.31}$	$12.25_{\pm 1.91}$	$45.61_{\pm 2.42}$	$50.04_{\pm 3.46}$	$9.94_{\pm 2.35}$	$0.29_{\pm 3.43}$	$22.31_{\pm 2.57}$	$19.14_{\pm 4.16}$
Ours								
HierRTL	$\mathbf{29.53}_{\pm 1.00}$	$\mathbf{26.93}_{\pm 0.91}$	$\mathbf{46.88}_{\pm 1.11}$	$\mathbf{46.37}_{\pm 1.51}$	$\mathbf{30.65}_{\pm 1.74}$	$\mathbf{33.06}_{\pm 8.39}$	$\mathbf{25.69}_{\pm 4.10}$	$\mathbf{28.72}_{\pm 1.94}$

Table 3: Focused ablations of Stage-I PPA and slack prediction and Stage-II region ranking.

Model	Stage-I: Design and Register Prediction						Stage-II: Region Ranking			
	Area ↓	Power ↓	WNS ↓	TNS ↓	PPA Avg. ↓	Slack ↓	$\rho \uparrow$	$\tau \uparrow$	Top-1 ↑	Norm. Regret ↓
(a) Hierarchy Construction										
w/o cone	4.75	5.83	5.41	6.68	5.67	3.87	25.90	23.85	42.79	50.55
w/o cone & design	5.65	7.35	6.63	7.62	6.81	7.35	20.63	18.06	39.19	53.29
(b) Topology-Only Hierarchies										
Fixed-hop topology	4.74	6.31	5.27	5.99	5.58	4.02	21.40	18.89	41.18	51.77
METIS topology	4.77	6.43	5.65	6.95	5.95	4.16	22.23	20.25	41.04	52.41
(c) Inter-Level Propagation and Fusion										
w/o upward PNA propagation	4.45	**5.35**	5.41	5.55	**5.19**	3.73	19.42	17.74	41.11	51.95
w/o decoder residual fusion	4.64	6.04	5.51	6.31	5.63	3.76	25.94	23.77	44.08	48.67
HierRTL	**4.32**	6.21	**5.07**	**5.43**	5.26	**3.59**	**29.53**	**26.93**	**46.88**	**46.37**

lowers ranking Spearman to 25.94. Thus, bidirectional exchange matters most for transferring hierarchical context to region ranking, even when its design-level regression effect is mixed.

5 Conclusion

We presented HierRTL, a hierarchical Graph U-Net that organizes RTL into node, register-cone, and design scales and exchanges information in both directions across this hierarchy. The resulting decoded node states support four design-level PPA heads and register-level setup-slack prediction within a single jointly supervised model. Across the completed structure-based comparisons, the model improves design-level PPA and register-slack NMAE while revealing that correlation and worst-register localization remain distinct challenges.

We further used bounded, equivalence-verified rewrites to define a soft opportunity for each editable region. A listwise downstream objective ranks regions within the same original design from its baseline graph, with region membership used only at readout. This reframes local optimization as an actionable pre-search decision: prioritize regions that are more likely to contain higher-value opportunities when rewrite, synthesis, and verification budgets are limited, without requiring a candidate graph or baseline PPA as input. The model does not generate rewrites or guarantee the gain of a particular unseen candidate, and it is not a replacement for signoff timing analysis; rather, it provides a structural prior for early PPA estimation and optimization triage. Improving robustness across RTL sources and validating the ranking in a closed optimization loop remain important directions for future work.

REPRODUCIBILITY STATEMENT

The model and evaluation protocols are described in the main text, with training configurations and checkpoint selection detailed in Appendix A.2. We intend to make the implementation code publicly available with the final version of this paper.

AI USE STATEMENT

In this work, we used generative AI tools for the following required research tasks: generating bounded RTL rewrite candidates for offline data construction, assisting with experimental methodology and design, and cleaning and reformatting the dataset. We have not used generative AI to develop the conceptual or theoretical model, formulate mathematical claims, refine research hypotheses, implement the core method, or interpret experimental results. The remaining required tasks—providing critical ingredients for proofs, assisting with proof writing, translation, and qualitative or thematic data analysis—were not applicable. In particular, generative AI was not used to verify data reliability: each retained candidate was independently checked through deterministic syntax checks, formal equivalence verification, and the fixed EDA evaluation flow. Additionally, we used generative AI tools for brainstorming and identifying relevant literature. Brainstorming outputs were treated only as suggestions; literature claims and bibliographic details were checked against the original papers and official sources. All AI-assisted outputs were reviewed by the authors, who take responsibility for the final paper and artifacts.

REFERENCES

Uri Alon and Eran Yahav. On the bottleneck of graph neural networks and its practical implications. *arXiv preprint arXiv:2006.05205*, 2020.

Robert Brayton and Alan Mishchenko. Abc: An academic industrial-strength verification tool. In *International Conference on Computer Aided Verification*, pp. 24–40. Springer, 2010.

Zhe Cao, Tao Qin, Tie-Yan Liu, Ming-Feng Tsai, and Hang Li. Learning to rank: from pairwise approach to listwise approach. In *Proceedings of the 24th international conference on Machine learning*, pp. 129–136, 2007.

Gabriele Corso, Luca Cavalleri, Dominique Beaini, Pietro Liò, and Petar Veličković. Principal neighbourhood aggregation for graph nets. *Advances in neural information processing systems*, 33:13260–13271, 2020.

Wenji Fang, Yao Lu, Shang Liu, Qijun Zhang, Ceyu Xu, Lisa Wu Wills, Hongce Zhang, and Zhiyao Xie. Masterrtl: A pre-synthesis ppa estimation framework for any rtl design. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–9. IEEE, 2023.

Wenji Fang, Shang Liu, Hongce Zhang, and Zhiyao Xie. Annotating slack directly on your verilog: Fine-grained rtl timing evaluation for early optimization. In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, pp. 1–6, 2024.

Wenji Fang, Shang Liu, Jing Wang, and Zhiyao Xie. Circuitfusion: multimodal circuit representation learning for agile chip design. In *International Conference on Learning Representations*, volume 2025, pp. 51523–51545, 2025a.

Wenji Fang, Shang Liu, Hongce Zhang, and Zhiyao Xie. A self-supervised, pre-trained, and cross-stage-aligned circuit encoder provides a foundation for various design tasks. In *Proceedings of the 30th Asia and South Pacific Design Automation Conference*, pp. 505–512, 2025b.

Hongyang Gao and Shuiwang Ji. Graph u-nets. In *international conference on machine learning*, pp. 2083–2092. PMLR, 2019.

George Karypis and Vipin Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Zeju Li, Changran Xu, Zhengyuan Shi, Zedong Peng, Yi Liu, Yunhao Zhou, Lingfeng Zhou, Chengyu Ma, Jianyuan Zhong, Xi Wang, et al. Deepcircuitx: A comprehensive repository-level dataset for rtl code understanding, generation, and ppa analysis. In *2025 IEEE International Conference on LLM-Aided Design (ICLAD)*, pp. 204–211. IEEE, 2025.

Shang Liu, Wenji Fang, Yao Lu, Jing Wang, Qijun Zhang, Hongce Zhang, and Zhiyao Xie. Rtlcoder: Fully open-source and efficient llm-assisted rtl code generation technique. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 44(4):1448–1461, 2024.

Yi Liu, Changran Xu, Yunhao Zhou, Zeju Li, and Qiang Xu. Deeprtl: Bridging verilog understanding and generation with a unified representation model. *arXiv preprint arXiv:2502.15832*, 2025a.

Yi Liu, Hongji Zhang, Yiwen Wang, Dimitris Tsaras, Lei Chen, Mingxuan Yuan, and Qiang Xu. Beyond tokens: Enhancing rtl quality estimation via structural graph learning. *arXiv preprint arXiv:2508.18730*, 2025b.

Yi Liu, Hongji Zhang, Yunhao Zhou, Zhengyuan Shi, Changran Xu, and Qiang Xu. Deeprtl2: A versatile model for rtl-related tasks. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 6485–6500, 2025c.

Yi Liu, Hongji Zhang, Lei Chen, Mingxuan Yuan, and Qiang Xu. Unirtl: Unifying code and graph for robust rtl representation learning. *arXiv preprint arXiv:2605.31040*, 2026.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

Ruiyang Ma, Yunhao Zhou, Yipeng Wang, Yi Liu, Zhengyuan Shi, Ziyang Zheng, Kexin Chen, Zhiqiang He, Lingwei Yan, Gang Chen, et al. Dynamicrtl: Rtl representation learning for dynamic circuit behavior. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 836–843, 2026.

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.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Clifford Wolf et al. Yosys open synthesis suite, 2016.

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

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems*, 31, 2018.

Yongan Zhang, Zhongzhi Yu, Yonggan Fu, Cheng Wan, and Yingyan Celine Lin. Mg-verilog: Multi-grained dataset towards enhanced llm-assisted verilog generation. In *2024 IEEE LLM Aided Design Workshop (LAD)*, pp. 1–5. IEEE, 2024.

Mingyu Zhao, Xun He, Jiawei Liu, Jianwang Zhai, and Chuan Shi. Topology matters in rtl circuit representa-tion learning. In *The Fourteenth International Conference on Learning Representations (ICLR)*, 2026.

Y Zhao et al. Codev: Empowering llms for verilog generation through multi-level summarization, version: 4, jul. 20. *arXiv preprint arXiv:2407.10424*, 2024.

A Additional Experimental Details

A.1 Detailed dataset statistics and processing

The two tasks share the original-design split whenever the Stage-I encoder is transferred to Stage II. Table 4 provides the exact accounting; Table 5 reports source and graph-size variation. Source tokens are descriptive statistics only and are not supplied to the graph models. The register annotations contain 164,419 endpoints, of which 151,305 have valid slack values and contribute to masked supervision. The source collections are public RTL corpora and model benchmarks; the corresponding dataset papers should be cited for DeepCircuitX, DeepRTL, DynamicRTL, MG-Verilog, and RTLCoder (Li et al., 2025; Liu et al., 2025a; Ma et al., 2026; Zhang et al., 2024; Liu et al., 2024).

Table 4: Dataset accounting after deterministic filtering and design-level splitting. "Regions" are baseline region targets; "rewrites" are retained equivalence-verified candidates used only to derive Stage II opportunity labels.

Task	Split	Designs	Regions	Rewrites	Valid slack labels
Stage I	Train	18,270	–	–	121,656
	Valid	2,272	–	–	14,763
	Test	2,306	–	–	14,886
	Total	22,848	–	–	151,305
Stage II	Train	1,547	4,794	17,009	–
	Valid	170	623	2,469	–
	Test	202	676	2,436	–
	Total	1,919	6,093	21,914	–

Table 5: Descriptive statistics of the Stage-I cohort. Size columns report P50/P95, i.e., the median and 95th percentile across designs within each source (or the full cohort in the last row); graph nodes and edges are from the typed implementation graph, and registers are counted before slack masking.

Source	Designs	RTL lines	RTL tokens	Nodes	Edges	Registers	Valid slack
DeepCircuitX	2,988	78/522	425/3,037	64/623	111/1,303	6/38	31,915
DeepRTL-v2	3,481	50/143	272/736	35/218	54/507	4/17	21,088
DynamicRTL	9,049	66/375	356/2,137	48/351	76/694	4/22	64,680
MG-Verilog	2,325	40/97	259/559	34/192	53/454	4/15	13,365
RTLCoder	5,005	44/113	187/518	20/130	30/278	2/13	20,257
All designs	22,848	54/285	284/1,622	37/325	59/680	4/21	151,305

Canonical graph and implementation labels. Each RTL design is parsed and elaborated with Yosys (Wolf et al., 2016); processes and memories are lowered, hierarchy is flattened, and unused logic is removed before exporting the graph. The resulting directed typed graph contains operator, port, constant, and register nodes, with data/control edges and bit-width attributes. Register-centered cones are constructed by reverse traversal of data and control dependencies, and a typed coarse graph is derived from register dependencies. Generic GNN controls use the node graph, whereas EDA baselines retain their own input representation.

Area, Power, WNS, and TNS labels are obtained after RTL normalization, optimized Design Compiler synthesis, and Innovus pre-route analysis under common clock, load, activity, and Nangate45-library settings. Register slack is aligned through the RTL-to-implementation register mapping; unmatched or invalid endpoints remain masked. Area and Power use training-only logarithmic transforms, WNS/TNS use signed logarithmic transforms, and register slack uses the same signed transform followed by training-set standardization.

Verified rewrite regions and opportunity labels. An AST-based parser identifies bounded editable units, including complete process blocks with traced fan-in, output cones, resolvable instance cones, and structural wrappers. For each selected region, an agent proposes replacement source for the marked span while preserving interfaces, clocks, resets, state updates, widths, signedness, and written signals. Each candidate is checked for source/AST consistency and functional equivalence against the immutable original RTL; failed, unknown, timed-out, or counterexample-producing checks are discarded. Accepted candidates are evaluated against the unchanged baseline with the same implementation flow. Candidate gains are clipped below at zero, combined with a no-

rewrite zero option, and converted into a train-only scaled soft-oracle target for each region and PPA objective. Candidate RTL and candidate graphs are not model inputs at inference. The equivalence-checking convention follows standard combinational/sequential equivalence practice (Brayton & Mishchenko, 2010).

In the released data construction, an agentic GPT-5.6 Luna model proposes the bounded replacements. It receives the complete original RTL as read-only context together with one explicitly writable source span, returns only a replacement for that span, and uses verifier and PPA feedback to make subsequent proposals. In the main construction, this feedback is produced by the commercial synthesis/implementation flow used to obtain the final labels. These model-assisted proposals are used only to construct the offline supervision set; the trained predictor does not call the language model at inference.

A.2 Training hyperparameters and selection protocol

Table 6 records the hyperparameters used by the reported mainline experiments. Unless a row explicitly says otherwise, all models use AdamW, FP32 arithmetic, two message-passing layers, hidden width 128, dropout 0.1, weight decay 10^{-4}, and independent heads for the four PPA targets. We report the number of random seeds rather than treating a single checkpoint as a distributional estimate. Validation is used for every checkpoint decision; Test is evaluated only after a selection manifest has been locked. AdamW follows decoupled weight decay (Loshchilov & Hutter, 2017).

Table 6: Training configurations. The graph and text baselines use the same optimizer conventions unless their input encoder is explicitly frozen.

Experiment	LR (encoder / head)	Budget / stopping	Loss / transforms	Batch / limits	Seeds
Stage I HierRTL joint	3×10^{-4} / 10^{-3} (PPA, slack)	400 ep.; prefix at 100/200/300/400	$\frac{1}{4}\sum_m \mathrm{MSE}_m + \mathrm{MSE}_{\mathrm{slack}}$; log/signed-log1p/train-z	100k nodes; 300k edges; 2M links; 32 designs/batch	2
Stage I PPA-only control	3×10^{-4} / 10^{-3}	400 ep.; validation prefix-best	Four independent PPA MSE heads; train-only transforms	Same as above	2
Generic GNN controls (GCN/GAT/GIN)	3×10^{-4} / 10^{-3}	400 ep.; validation prefix-best	Joint PPA/slack; node-graph readout	Same limits/dropout	3
Stage II region ranking	3×10^{-4} / 10^{-3} (encoder / rank head)	100 ep.; min 10; patience 12; $\Delta = 10^{-4}$	Soft-oracle/listwise ($T = 0.25$); 1-layer, 4-head attention	128 region tokens; hidden 128; dropout 0.1	3

For Stage I, the four PPA targets are trained as separate heads so that a large-valued target cannot dominate the shared representation. Area and Power are transformed with a training-only logarithm; signed WNS/TNS use signed $\log(1 + |x|)$; slack uses the same signed transform followed by a training-only mean and standard deviation. The local slack loss is averaged within each design before the batch average, and masked endpoints contribute neither loss nor metric. The long-horizon runs disable early termination and retain the best validation checkpoint in each prefix; the reported mainline is the validation-selected prefix-400 checkpoint.

For Stage II, the selected Stage-I checkpoint initializes the encoder and the rank head is trained end to end. The soft-oracle target is a temperature- scaled expectation over verified candidate opportunities and the baseline zero option. Listwise normalization is performed separately within each original design and target; ties are retained rather than broken by row order. The region indicator is consumed only by the final readout. The three Stage-II seeds are independent downstream initializations, while the source encoder checkpoint is held fixed across those runs; consequently the reported spread does not include Stage-I pretraining randomness.