R2G: A Multi-View Circuit Graph Benchmark Suite from RTL to GDSII
Supplementary Material

This supplementary material provides comprehensive details, implementation specifics, and additional experimental analyses to support the main manuscript. The document is organized as follows: **Section A** offers a practical guide and API examples for utilizing the R2G benchmark suite. **Section B** categorizes existing graph datasets to contextualize the unique contribution of the R2G benchmark. **Section C** elaborates on the data processing pipeline, specifically the DEF parser used to translate physical layouts into graph structures. **Section D** presents granular statistics across the multi-view graph representations. **Section E** provides an in-depth look at the experimental setup, detailing hyperparameter tuning strategies. Finally, **Section F** presents ablation studies on GNN and head depths.

A. Usage

R2G provides a reproducible pipeline that converts OpenROAD DEF files into multi-view circuit graphs and supports standardized GNN training and evaluation. The workflow consists of two main stages: (1) graph generation from physical-design files and (2) model training and evaluation on the generated datasets.

A.1. Data Generation Pipeline

R2G converts physical design outputs into graph datasets through a multi-stage pipeline. Starting from DEF files generated by the OpenROAD flow, the framework constructs typed circuit graphs and produces datasets compatible with PyTorch Geometric.

The pipeline consists of three steps:

- **Heterograph generation:** DEF files are parsed to construct typed heterogeneous graphs containing gates, nets, pins, and IO nodes. Multiple graph views (B–F) can be generated using the provided scripts in `data_pipeline/heterograph_generation/`.
- **Homograph conversion:** The heterogeneous graphs are converted into homogeneous graphs for efficient GNN training using the converters in `data_pipeline/homograph_conversion/`.
- **Graph merging:** Graphs from multiple designs are merged into a single dataset to enable large-scale training. The resulting merged graphs are stored as PyTorch `.pt` files and serve as the final benchmark datasets.

This pipeline converts DEF-based physical layouts into standardized circuit graphs suitable for machine learning tasks such as placement and routing prediction.

A.2. Model Training

The generated datasets can be used directly for GNN training. R2G provides training scripts for prediction tasks at different supervision granularities.

- **Node-centric configurations:** Training scripts are located in the `gnn-node/` directory. In the released baselines, they are used for placement-oriented targets such as HPWL.
- **Edge-centric configurations:** Training scripts are located in the `gnn-edge/` directory. In the released baselines, they are used for routing-oriented targets such as wire-length prediction.

Each training pipeline includes dataset loading, feature encoding, neighbor sampling, model initialization, and evaluation. The framework supports multiple GNN architectures including GINE, ResGatedGCN, and GAT.

A.3. Training Example

A typical node-centric configuration for the placement task is shown below:

```
cd gnn-node

python main.py \
    --dataset place_B_homograph \
    --task_level node \
    --task regression \
    --model gine \
    --num_gnn_layers 4 \
    --num_head_layers 4 \
    --hid_dim 256 \
    --epochs 100 \
    --gpu 0
```

Listing 1. Example training command for a node-centric placement configuration.

Similarly, routing tasks can be trained using the scripts in `gnn-edge/`.

A.4. Evaluation

During training, the framework automatically reports standard regression metrics including Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and the coefficient of determination (R^2). The best-performing model on the validation set is saved and evaluated on the test set.

Additional analysis tools are provided to visualize prediction quality, including scatter plots and label distribution statistics.

Table 9. Overview of existing graph datasets. Tasks include Classification (C), Regression (R), and Link prediction (L).

Benchmark Suite		Category	Level	Task	Statistics			Metric
					#Node/G	#Edge/G	#Degree/G	
OGB	ogbn-products	Product	node	C	2,449,029	61,859,140	50.5	Accuracy
	ogbn-mag	Academic	node	C	1,939,743	21,111,007	21.77	Accuracy
	ogbl-ppa	Bio	edge	L	576,289	30,326,273	105.2	Hits@100
	ogbl-citation2	Citation	edge	L	2,927,963	30,561,187	20.9	MRR
TU	QM9	Molecules	node	R	18	19	2.07	MAE
	ZINC	Molecules	graph	R	23	25	2.14	MAE
	PROTEINS	Bio	graph	C	39	73	3.73	Accuracy
	reddit	Social	graph	C	24	25	2.03	F1-Score
	COLLAB	Social	graph	C	74	2,458	37.39	F1-Score
SNAP	ego-Twitter	Social	node	C	81306	1768149	43.49	F1-Score
	com-Youtube	Social	node	C	1134890	2987624	5.27	F1-Score
	cit-Patents	Citation	node	C	3774768	16518948	8.75	F1-Score
	web-Google	Web	node	C	875713	5105039	11.66	F1-Score
	amazon-meta	Product	node	C	548552	1788725	6.52	F1-Score
R2G	B_graph	Circuit Design	node	R	124408	149287	2.4	MAE
	C_graph	Circuit Design	node	R	49921	110166	4.41	MAE
	D_graph	Circuit Design	edge	R/C	89640	107689	2.4	MAE/F1-Score
	E_graph	Circuit Design	edge	R/C	64975	78789	2.43	MAE/F1-Score
	F_graph	Circuit Design	edge	R/C	25061	46597	3.72	MAE/F1-Score

B. Overview of existing graph datasets

Benchmarks and tasks. We situate widely used graph suites alongside our circuit-graph benchmark, as summarized in Table 9. OGB [13] targets large-scale node, link, and graph tasks with unified splits and metrics; TUDataset [21] aggregates small-to-mid collections (molecules, bio, social) for graph-level classification and regression; SNAP [17] curates massive social, web, and citation networks for node classification and link prediction. These domain-agnostic, predominantly single-view resources lack typed heterogeneity, multi-terminal hyperedges, and geometry-aware attributes required for physical design.

R2G differs fundamentally. It releases typed, heterogeneous circuit graphs extracted from DEF, with stage-aware, multi-view representations and *information parity*. Targets are attached at the entity resolution induced by each view, with domain metrics aligned to EDA objectives in placement and routing. Graphs are industrial yet tractable, capturing net topology, pin-level connectivity, and macro-to-cell context. With scalable loaders, unified splits, metrics, and reproducible baselines, R2G enables fair cross-view comparison and cleanly decouples representation from modeling across late physical-design stages.

C. DEF Parser

To bridge the gap between physical layout implementation and graph-based learning, the R2G framework incorporates a specialized DEF parser. This component is responsible for extracting semantic and geometric information from standard Design Exchange Format (DEF) files, converting them into the structured graph representations used in our benchmark. In this section, we first detail the structural evolution of DEF files throughout the design flow to clarify the data availability at each stage. Subsequently, we outline the algorithmic procedure used to translate these raw descriptions into heterogeneous graphs.

C.1. DEF File Description

The Design Exchange Format (DEF) serves as the standard for exchanging physical layout information. Its content evolves significantly through the physical design stages:

- **Floorplan Stage:** This stage establishes the "geometric stage" and power distribution. The DEF file defines the design bounds (`DIEAREA`), standard cell rows (`ROW`), and routing tracks (`TRACKS`). Power networks (`SPECIALNETS`) are often generated here (e.g., VDD/VSS stripes). Logical connections (`NETS`) exist but lack physical geometry, and components generally lack fixed coordinates unless manually locked.
- **Placement Stage:** Specific physical coordinates are assigned. Standard cells in the `COMPONENTS` section are updated with `+ PLACED (x y)` coordinates and orientations. Similarly, I/O pins in the `PINS` section are assigned specific metal layers and physical locations (`+ PORT + LAYER`). Signal nets remain logical connections without detailed routing segments.
- **Routing Stage:** The layout is finalized with detailed interconnects. A global routing grid (`GCELLGRID`) is often added. The `NETS` section is populated with detailed physical paths, including wire segments, turns, and cross-layer vias using the `+ ROUTED ... NEW` syntax. The component count may increase due to the insertion of filler cells or buffers.

Listing 2 illustrates a routed DEF file structure, highlighting the information accumulated from these stages.

C.2. DEF-to-Graph Conversion

The conversion process from a raw DEF file to a structured graph representation is outlined in Algorithm 1.

The DEF-to-Graph Conversion Algorithm transforms a Design Exchange Format (DEF) file, which describes the physical layout of an integrated circuit, into a heterogeneous graph representation suitable for machine learning and analysis tasks. This algorithm systematically converts circuit components into graph nodes and their interconnections into graph edges, creating a structured representation that preserves the topological and functional relationships within the circuit design.

The conversion process involves three main phases: Node Construction (creating nodes for gates, nets, and pins), Edge Construction (establishing connectivity between components), and Feature Encoding (adding numerical attributes to nodes and edges). The resulting heterogeneous graph captures both the structural hierarchy and physical properties of the circuit, enabling applications in optimization, verification, and AI-driven design automation.

Table 10. Comparison of graph statistics across graph views and design categories.

Cat.	RTL2G	#nodes					#edges					avg_degree					avg_shortest path				
		(b)	(c)	(d)	(e)	(f)	(b)	(c)	(d)	(e)	(f)	(b)	(c)	(d)	(e)	(f)	(b)	(c)	(d)	(e)	(f)
A/V ctrl.	ss_pcm	2.44k	1.01k	1.89k	1.46k	0.46k	2.88k	2.02k	2.36k	1.66k	1.01k	2.37	4.01	2.49	2.28	4.35	12.07	3.00	9.81	12.69	4.05
	ac97_ctrl	58.22k	22.99k	40.86k	30.63k	10.36k	70.60k	47.20k	52.19k	36.83k	23.50k	2.43	4.11	2.55	2.40	4.54	13.41	3.00	13.76	21.20	5.65
	vga_lcd	531.33k	206.01k	369.72k	275.41k	94.51k	650.84k	445.88k	455.43k	325.55k	197.12k	2.45	4.33	2.46	2.36	4.17	14.33	3.02	24.37	31.42	8.00
	Average	197.33k	76.67k	137.49k	102.50k	35.11k	241.44k	165.03k	170.00k	121.35k	74.08k	2.42	4.15	2.50	2.35	4.35	13.27	3.01	15.98	21.77	5.90
Crypto Core	des3_area	9.78k	3.65k	7.94k	6.33k	1.80k	12.46k	9.29k	11.17k	9.41k	5.10k	2.55	5.09	2.81	2.97	5.66	18.16	3.28	15.43	12.42	6.13
	systemcdes	14.17k	5.76k	10.73k	8.14k	2.79k	17.02k	12.84k	13.48k	10.13k	5.73k	2.40	4.46	2.51	2.49	4.11	20.60	3.18	19.62	17.26	7.20
	systemcaes	41.08k	16.13k	31.74k	24.34k	7.78k	50.29k	36.81k	40.30k	30.46k	16.83k	2.45	4.56	2.54	2.50	4.33	18.02	3.16	13.91	18.13	6.20
	sha256	62.83k	25.06k	47.57k	36.52k	11.83k	76.32k	56.36k	59.51k	45.47k	24.70k	2.43	4.50	2.50	2.49	4.18	18.66	2.88	16.43	17.12	6.75
	aes_secworks	128.40k	46.60k	96.64k	75.04k	22.12k	164.11k	113.16k	128.92k	99.87k	56.87k	2.56	4.86	2.67	2.66	5.14	19.92	2.93	19.73	20.61	7.07
	aes_xcrypt	165.76k	58.42k	135.12k	107.08k	28.43k	215.07k	156.35k	188.34k	156.69k	81.88k	2.60	5.35	2.79	2.93	5.76	20.68	2.97	17.39	16.57	7.17
	Average	73.85k	26.27k	58.29k	44.58k	12.46k	92.55k	64.13k	75.32k	58.67k	31.84k	2.50	4.77	2.64	2.64	4.86	19.34	3.07	17.08	17.02	6.75
Processor	tv80	31.14k	11.44k	24.91k	19.47k	5.51k	39.45k	29.34k	34.25k	28.48k	15.14k	2.53	5.13	2.75	2.93	5.50	16.71	3.10	14.48	13.95	5.94
	tv80s	37.24k	14.39k	28.88k	21.96k	6.97k	45.74k	35.41k	37.09k	29.41k	15.52k	2.46	4.92	2.57	2.68	4.45	19.02	3.14	17.64	17.13	7.05
	riscv32i	53.23k	19.76k	40.78k	31.50k	9.41k	66.91k	47.74k	56.38k	45.79k	26.04k	2.51	4.83	2.77	2.91	5.53	17.43	3.20	15.24	15.57	5.42
	ibex	96.60k	35.64k	72.50k	56.14k	16.62k	122.18k	84.69k	98.88k	79.63k	43.94k	2.53	4.75	2.73	2.84	5.29	18.44	3.05	17.90	18.85	6.86
	tinyRocket	50.63k	32.60k	42.82k	15.43k	27.66k	36.27k	66.69k	26.03k	13.54k	14.90k	1.43	4.09	1.22	1.75	1.08	11.00	2.40	9.68	12.07	4.53
	swerv	538.36k	196.94k	384.55k	294.23k	92.36k	684.89k	469.98k	510.70k	388.58k	229.49k	2.54	4.77	2.66	2.64	4.97	17.10	3.08	19.33	28.16	8.27
	bp_multi	296.72k	171.27k	235.98k	107.09k	130.34k	250.15k	353.77k	176.42k	102.51k	86.96k	1.69	4.13	1.50	1.91	1.33	11.72	2.58	18.45	25.72	6.42
	Average	154.85k	69.63k	118.58k	78.52k	29.88k	177.80k	155.03k	163.18k	114.28k	61.65k	2.15	4.51	2.17	2.41	3.90	15.90	2.94	16.10	18.64	6.36
Comm. ctrl.	uart	2.73k	1.11k	1.94k	1.48k	0.52k	3.29k	2.30k	2.60k	1.88k	1.15k	2.41	4.13	2.67	2.54	4.47	13.80	3.27	14.79	10.61	5.63
	sasc_top	3.45k	1.38k	2.44k	1.84k	0.63k	4.17k	2.88k	3.01k	2.12k	1.33k	2.42	4.18	2.47	2.31	4.22	12.20	2.88	12.69	15.79	4.75
	i2c_verilog	4.69k	1.85k	3.39k	2.56k	0.87k	5.70k	4.03k	6.54k	5.28k	4.12k	2.43	4.35	3.86	4.13	9.51	13.42	3.02	11.96	13.93	5.49
	simple_spi_top	4.57k	1.84k	3.35k	2.52k	0.85k	5.49k	3.89k	4.17k	3.01k	1.80k	2.40	4.23	2.49	2.39	4.23	13.33	2.90	12.23	13.55	4.93
	spi_top	15.62k	6.07k	12.31k	9.47k	2.93k	19.18k	14.21k	19.15k	15.39k	9.99k	2.46	4.68	3.11	3.25	6.81	16.79	3.02	13.75	14.99	5.68
	dynamic_node	67.87k	26.28k	46.91k	35.50k	12.10k	83.87k	56.29k	57.47k	40.55k	24.88k	2.47	4.28	2.45	2.28	4.11	15.10	3.22	16.09	15.00	6.62
	pci	72.69k	27.53k	48.45k	36.59k	12.23k	90.70k	56.01k	72.86k	51.99k	38.15k	2.50	4.07	3.01	2.84	6.24	15.13	2.88	16.35	25.42	6.37
	usb_funct	67.30k	26.63k	48.60k	36.55k	12.30k	81.58k	58.85k	60.03k	44.10k	23.78k	2.42	4.42	2.47	2.41	3.87	21.41	3.33	19.13	21.79	7.43
	wb_dma_top	19.52k	8.22k	14.60k	11.08k	3.95k	23.03k	16.72k	17.79k	12.52k	7.62k	2.36	4.07	2.44	2.26	3.86	15.38	3.17	13.09	15.66	5.76
	wb_conmax	181.45k	71.84k	144.40k	110.71k	36.24k	221.76k	177.04k	183.96k	145.80k	76.57k	2.44	4.93	2.55	2.63	4.23	24.35	2.96	21.46	18.56	9.04
	usb_phy	2.73k	1.13k	1.84k	1.36k	0.51k	3.22k	2.35k	2.14k	1.42k	0.91k	2.36	4.15	2.33	2.09	3.56	10.67	3.27	11.10	11.89	4.41
	Average	42.20k	16.68k	31.93k	24.40k	8.76k	50.89k	39.33k	43.32k	33.03k	19.54k	2.42	4.30	2.67	2.61	4.78	15.65	3.06	14.71	16.23	5.98
DSP Core	fir	22.63k	8.82k	16.90k	12.98k	3.98k	27.68k	19.51k	21.46k	16.28k	8.80k	2.45	4.42	2.54	2.51	4.42	17.94	3.07	16.16	20.45	6.23
	jpeg	331.18k	134.01k	244.06k	183.26k	60.85k	394.38k	301.36k	297.27k	236.65k	118.01k	2.38	4.50	2.44	2.58	3.88	23.11	3.04	23.11	24.50	8.91
	idft	817.88k	313.24k	527.42k	392.61k	134.94k	1009.40k	622.01k	590.78k	382.70k	236.06k	2.47	3.97	2.24	1.91	3.50	13.06	3.26	19.57	26.46	6.64
	Average	390.56k	152.02k	262.79k	196.28k	66.59k	477.15k	314.29k	303.17k	211.88k	120.96k	2.43	4.30	2.41	2.33	3.93	18.04	3.12	19.61	23.80	7.26
Total Average		124.41k	49.92k	89.64k	64.98k	25.06k	149.29k	110.17k	107.69k	78.79k	46.60k	2.40	4.48	2.55	2.56	4.58	16.43	3.04	16.16	18.25	6.35

D. Statistics of Multi-Views

This appendix summarizes structural statistics for 30 circuits across five graph views (b–f): #nodes, #edges, avg_degree, and avg_shortest path computed by a unified late-stage pipeline. Table 10 shows consistent patterns aligned with physical design: view (b) has *moderate* degree (≈2.4–2.6) and *long* paths (≈11–23); view (c) is the *densest* node-centric option (higher degree, very short paths); views (d)/(e) retain *moderate* degrees with *mid-to-long*/*very long* diameters; view (f) is *dense, small-world* (high degree, short paths). These structures match trends in Table 3 and Table 4.

D.1. Why view (b) leads and node-centric views prevail

D.1.1. Average Statistics

view (b) balances locality and reach: *moderate degree* (≈2.40) with *long paths* (≈16.43). A moderate degree constrains neighborhood breadth—reducing noise amplification and over-mixing—while long paths preserve global dependencies needed for macro-to-cell interactions and net connectivity. view (c) is the *densest* node-centric option (degree ≈4.48, paths ≈3.04): high degree and very short paths concentrate information locally; this aids fine-grained aggregation but compresses structural distance, limiting long-range context when global coordination is required. views (d)/(e) retain *moderate degrees* (≈2.55/2.56) and *mid-to-long paths* (≈16.16/18.25), reflecting pin-level dependencies along nets: multi-hop propagation is supported without saturating neighborhoods. view (f) is *dense, small-world* (degree ≈4.58, paths ≈6.35), tying distant subcircuits through hubs; this increases correlation but can blur constraint boundaries and inject hub bias.

D.1.2. Cross-Stage Outcomes

Placement favors node-centric graphs: view (b) is strongest, with view (c) close. Placement decisions rely on global context and net topology; B's long paths and controlled degree deliver broad context without noise, explaining its margin. C competes when local density helps, but its short-path bias can hamper cases needing global coordination. In routing, view (b) remains strongest overall, while among edge-centric views, views (d)/(e) are competitive: their mid-to-long paths track channel capacity and congestion cascades along connections, making them useful complements for routing-oriented analysis. view (f) underperforms because hub-dominated mixing erodes locality and misaligns with edge constraints.

D.1.3. Summaries

Prefer view (b) as a robust default across stages. Use view (c) judiciously for workloads dominated by local interactions; treat views (d)/(e) as complementary choices when

```
DESIGN ac97_top ;
UNITS DISTANCE MICRONS 2000 ;
DIEAREA ( 0 0 ) ( 434390 434390 ) ;

# [Floorplan] Rows, Tracks, and Power Grid defined
ROW ROW_0 FreePDK45_38x28_10R_NP_162NW_34O 2280 2800 N
      DO 1131 BY 1 STEP 380 0 ;
TRACKS X 190 DO 1143 STEP 380 LAYER metal1 ;
SPECIALNETS 2 ;
    - VDD ( * VDD ) + USE POWER
      + ROUTED metal7 2800 + SHAPE STRIPE ( 61800
     396800 ) ( 398760 396800 ) ;
END SPECIALNETS

# [Routing] Global routing grid added
GCELLGRID X 0 DO 103 STEP 4200 ;

# [Placement] Coords added (+ PLACED); [Routing] Count
      increases
COMPONENTS 11178 ;
    - _10221_ CLKBUF_X2 + PLACED ( 169860 109200 ) FN
     ;
    - _10222_ INV_X1    + PLACED ( 165300 109200 ) FN
     ;
END COMPONENTS

# [Placement] Pin physical details (+ PORT + LAYER)
     added
PINS 132 ;
    - ac97_reset_pad_o_ + NET ac97_reset_pad_o_ +
     DIRECTION OUTPUT + USE SIGNAL
      + PORT
        + LAYER metal6 ( -140 -140 ) ( 140 140 )
        + PLACED ( 127870 140 ) N ;
END PINS

# [Routing] Detailed geometry segments (+ ROUTED ...
     NEW) added
NETS 12959 ;
    - _00000_ ( u10.dout[0]$_DFF_P_ D ) ( _11175_ ZN )
      + USE SIGNAL
      + ROUTED metal2 ( 91770 88060 ) ( * 96600 )
      NEW     metal2 ( 91770 96600 ) ( 92150 * )
      NEW     metal1 ( 91770 88060 ) via1_4 ;
END NETS

END DESIGN
```

Listing 2. A DEF file example showing the accumulation of data from Floorplan to Routing.

routing fidelity is prioritized. Avoid view (f) for headline reporting due to small-world mixing and hub bias that harm both placement coordination and routing locality.

D.2. Why view (d) leads among edge-centric views: statistical comparison

D.2.1. Key Statistics

view (d) combines *moderate degree* with *mid-to-long paths*, yielding expressive yet unsaturated neighborhoods. Compared with view (e), degrees are similar on average, but E's paths are consistently longer, shifting useful interactions to larger hop distances. view (f) exhibits *higher degrees* and *shorter paths*, encouraging small-world mixing and hub bias that blur fine-grained signals. In edges-to-nodes balance, view (d) remains near-linear, whereas view (f) increases density and alters neighborhood selectivity.

Algorithm 1 DEF-to-Graph Conversion Process

Require: DEF file $\mathcal{D}$
Ensure: Heterogeneous Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
Initialization: $\mathcal{V} \leftarrow \emptyset, \mathcal{E} \leftarrow \emptyset$
$\mathcal{D}_{data} \leftarrow \text{ParseDEF}(\mathcal{D})$
{**Phase 1: Node Construction**}
for all $comp \in \mathcal{D}_{data}.components$ **do**
  $v_{gate} \leftarrow \text{CreateGateNode}(comp)$
  $\mathcal{V}.add(v_{gate})$
end for
for all $net \in \mathcal{D}_{data}.nets$ **do**
  $v_{net} \leftarrow \text{CreateNetNode}(net)$
  $\mathcal{V}.add(v_{net})$
end for
for all $pin \in \mathcal{D}_{data}.pins$ **do**
  $v_{io} \leftarrow \text{CreateIOPinNode}(pin)$
  $\mathcal{V}.add(v_{io})$
end for
for all $int\_pin \in \mathcal{D}_{data}.internal\_pins$ **do**
  $v_{pin} \leftarrow \text{CreatePinNode}(int\_pin)$
  $\mathcal{V}.add(v_{pin})$
end for
{**Phase 2: Edge Construction**}
for all $net \in \mathcal{D}_{data}.nets$ **do**
  for all $(comp, pin) \in net.connections$ **do**
    if $comp ==$ 'PIN' **then**
      $\mathcal{E}.add((v_{io}, v_{net}))$ {Edge: IO Pin $\leftrightarrow$ Net}
    else
      $\mathcal{E}.add((v_{pin}, v_{net}))$ {Edge: Internal Pin $\leftrightarrow$ Net}
    end if
  end for
end for
for all $comp \in \mathcal{D}_{data}.components$ **do**
  for all $pin \in comp.pins$ **do**
    $\mathcal{E}.add((v_{gate}, v_{pin}))$ {Edge: Gate $\leftrightarrow$ Pin}
  end for
end for
{**Phase 3: Feature Encoding**}
for all $v \in \mathcal{V}$ **do**
  $v.x \leftarrow \text{EncodeNodeFeatures}(v)$
end for
for all $e \in \mathcal{E}$ **do**
  $e.attr \leftarrow \text{EncodeEdgeFeatures}(e)$
end for
$\mathcal{G}.global \leftarrow \text{ExtractGlobalFeatures}(\mathcal{D}_{data})$
return $\mathcal{G}$

D.2.2. Cross-Stage Alignment

For routing-oriented analysis, view (d) is the strongest edge-centric complement; view (e) supports longer-range effects when they are critical; view (f)'s small-world structure is generally misaligned.

D.2.3. Summaries

Average structural statistics explain stage-level outcomes without model-specific details: view (b) is robust across

Table 11. Placement Task Hyperparameters for Different Graph Views

Hyper	lr	wd	hid_dim	act	gnn_layer	dropout	head_layer	head_dim
(b)								
GINE	1e-4	1e-4	256	relu	4	0.3	2	256
ResGatedGCN	1e-4	1e-4	256	relu	4	0.3	2	256
GAT	1e-4	1e-4	256	relu	4	0.3	2	256
(c)								
GINE	1e-4	1e-4	256	relu	4	0.3	2	256
ResGatedGCN	1e-4	1e-4	256	relu	4	0.3	2	256
GAT	1e-4	1e-4	256	relu	4	0.3	2	256
(d)								
GINE	5e-4	1e-5	256	leakyrelu	4	0.3	2	256
ResGatedGCN	5e-4	1e-5	256	leakyrelu	4	0.3	2	256
GAT	5e-4	1e-5	256	leakyrelu	4	0.3	2	256
(e)								
GINE	5e-4	1e-5	256	leakyrelu	4	0.3	2	256
ResGatedGCN	5e-4	1e-5	256	leakyrelu	4	0.3	2	256
GAT	5e-4	1e-5	256	leakyrelu	4	0.3	2	256
(f)								
GINE	5e-4	1e-5	256	leakyrelu	4	0.3	2	256
ResGatedGCN	5e-4	1e-5	256	leakyrelu	4	0.3	2	256
GAT	5e-4	1e-5	256	leakyrelu	4	0.3	2	256

Table 12. Routing Task Hyperparameters for Different Graph Views

Hyper	lr	wd	hid_dim	act	gnn_layer	dropout	head_layer	head_dim
(b)								
GINE	1e-4	1e-4	256	leakyrelu	4	0.3	2	256
ResGatedGCN	1e-4	1e-4	256	leakyrelu	4	0.3	2	256
GAT	1e-4	1e-4	256	leakyrelu	4	0.3	2	256
(c)								
GINE	1e-4	1e-4	256	leakyrelu	4	0.3	2	256
ResGatedGCN	1e-4	1e-4	256	leakyrelu	4	0.3	2	256
GAT	1e-4	1e-4	256	leakyrelu	4	0.3	2	256
(d)								
GINE	5e-4	1e-5	256	leakyrelu	4	0.3	2	256
ResGatedGCN	5e-4	1e-5	256	leakyrelu	4	0.3	2	256
GAT	5e-4	1e-5	256	leakyrelu	4	0.3	2	256
(e)								
GINE	5e-4	1e-5	256	leakyrelu	4	0.3	2	256
ResGatedGCN	5e-4	1e-5	256	leakyrelu	4	0.3	2	256
GAT	5e-4	1e-5	256	leakyrelu	4	0.3	2	256
(f)								
GINE	5e-4	1e-5	256	leakyrelu	4	0.3	2	256
ResGatedGCN	5e-4	1e-5	256	leakyrelu	4	0.3	2	256
GAT	5e-4	1e-5	256	leakyrelu	4	0.3	2	256

placement and routing; view (c) is strong but less stable; views (d)/(e) are viable edge-centric complements; view (f) is not recommended for headline results.

E. Hyperparameter Tuning

The experimental results for the Placement Task across different graph views (b-f), as presented in Table 3 of the main paper, and for the Routing Task in Table 4, were obtained using the specific hyperparameter configurations detailed in Table 11 and Table 12, respectively. To ensure a fair and consistent comparison across the different graph views, each model (GINE, ResGatedGCN, GAT) utilized a single, fixed set of hyperparameters for all views within a given task. This means that the same hyperparameter values were applied to all views (b), (c), (d), (e), and (f) for the Placement task, and similarly for the Routing task, without further view-specific tuning.

F. GNN Depth and Head Depth

The following four tables (Table 13-Table 16) present supplementary experiments for the Placement and Routing tasks under view (d). These experiments investigate the impact of two key architectural hyperparameters—the depth of the Graph Neural Network (GNN) backbone and the number of prediction head layers—on model performance. The results indicate that the number of GNN layers has a relatively minor and inconsistent effect, whereas the number of head layers is a crucial factor significantly influencing the outcomes.

Table 13. Placement on view (d) vs. GNN depth (3–6).

	GINE				ResGatedGCN			
#layers	MAE↓	RMSE↓	R^2 ↑	#Param.	MAE↓	RMSE↓	R^2 ↑	#Param.
3	0.4552	0.6098	0.6811	0.83M	**0.4521**	**0.6026**	**0.6886**	1.62M
4	0.4451	0.5937	0.6975	1.03M	0.4670	0.6197	0.6704	2.08M
5	0.4837	0.6467	0.6414	1.23M	0.4700	0.6077	0.6833	2.54M
6	**0.4063**	**0.5478**	**0.7427**	1.42M	0.5449	0.6786	0.6051	3.00M

Table 14. Routing on view (d) vs. GNN depth (3–6).

	GINE				ResGatedGCN			
#layers	MAE↓	RMSE↓	R^2 ↑	#Param.	MAE↓	RMSE↓	R^2 ↑	#Param.
3	0.5470	0.7172	0.6000	0.83M	0.5732	0.7557	0.5558	1.62M
4	0.6111	0.7920	0.5122	1.03M	0.6007	0.7826	0.5237	2.08M
5	**0.5062**	**0.6805**	**0.6398**	1.22M	**0.5413**	**0.7091**	**0.6090**	2.54M
6	0.5964	0.7571	0.5542	1.42M	0.6803	0.8942	0.3782	3.00M

Table 15. Placement on view (d) vs. head layers (1–4).

	GINE				ResGatedGCN			
#layers	MAE↓	RMSE↓	R^2 ↑	#Param.	MAE↓	RMSE↓	R^2 ↑	#Param.
1	nan	nan	nan	0.90M	nan	nan	nan	1.95M
2	0.4451	0.5937	0.6975	1.03M	0.4670	0.6197	0.6704	2.08M
3	**0.4337**	**0.5804**	**0.7109**	1.10M	0.4576	0.6043	0.6866	2.15M
4	0.4359	0.5851	0.7062	1.16M	**0.4418**	**0.5922**	**0.6990**	2.21M

Table 16. Routing on view (d) vs. head layers (1–4).

	GINE				ResGatedGCN			
#layers	MAE↓	RMSE↓	R^2 ↑	#Param.	MAE↓	RMSE↓	R^2 ↑	#Param.
1	nan	nan	nan	0.90M	nan	nan	nan	1.95M
2	0.6111	0.792	0.5122	1.03M	0.6007	0.7826	0.5237	2.08M
3	**0.4637**	**0.6148**	**0.7060**	1.09M	0.5678	0.7551	0.5565	2.14M
4	0.4971	0.6636	0.6581	1.16M	**0.5445**	**0.7099**	**0.6089**	2.21M

Impact of GNN Depth (Table 13 and Table 14): The number of GNN layers (ranging from 3 to 6) does not exhibit a consistent impact on model performance. In Table 13 (Placement), GINE achieves its best results at deeper configurations, whereas ResGatedGCN performs best with fewer layers. A similar observation holds for routing in Table 14, where both models achieve their strongest results at intermediate depths rather than at the deepest settings. Across both tasks, the performance metrics fluctuate across layer configurations without showing a clear monotonic trend. These results suggest that increasing GNN depth alone does not consistently improve prediction quality, and that message-passing depth is not a dominant factor for these tasks.

Impact of Head Layers (Table 15 and Table 16): In contrast, the number of head layers (ranging from 1 to 4) has a much stronger influence on performance. In Table 15 (Placement), increasing the depth of the prediction head leads to noticeable improvements for both GINE and ResGatedGCN. A similar trend is observed in Table 16 (Routing), where deeper prediction heads consistently produce

better results. These improvements indicate that the design of the prediction head plays a crucial role in model performance. In particular, richer head architectures appear to provide stronger capacity for mapping learned node or edge embeddings to task-specific targets, highlighting the importance of prediction-head design in GNN-based physical design tasks.