CLIMBING THE DESIGN LADDER: SEQUENTIAL KNOWLEDGE DISTILLATION FOR EARLY-STAGE CIRCUIT TIMING PREDICTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Integrated circuit design involves multiple design stages: logic synthesis, floorplanning, placement, and routing, with each stage taking hours to weeks to complete. Discovering timing violations late in this flow forces costly iterations back to earlier stages, wasting computational resources and delaying product launches. While predicting post-routing timing from early-stage data could prevent these failures, existing machine learning approaches struggle with the massive abstraction gap between post-synthesis logical descriptions and post-routing physical layouts. We propose STEP-KD (Sequential Timing Evaluation via Progressive Knowledge Distillation), which leverages intermediate design stages as "stepping stones" for progressive knowledge transfer rather than attempting direct prediction. STEP-KD trains teacher models at the post-routing, post-placement, and post-floorplan stages, then sequentially distills their knowledge to a post-synthesis student model through representation alignment. Experiments on diverse circuits demonstrate that STEP-KD reduces timing prediction error compared to direct distillation and supervised baselines, and in most settings compared to the industry-standard Static Timing Analysis (STA) tool. STEP-KD reduces the weighted mean absolute percentage error of Total Negative Slack prediction to 19.78%, compared with 74.84% for STA. Our proposed method is step forward to identify timing problems earlier, avoiding expensive late-stage redesigns.

1 INTRODUCTION

The semiconductor industry underpins technological advances in AI/ML accelerators, autonomous vehicles, 5G/6G communications, and quantum computing. At the heart of this industry lies the integrated circuit (IC) design flow, which turns a register-transfer level (RTL) hardware description into a manufacturable layout through a sequence of stages: logic synthesis maps the RTL to a netlist of library cells, floorplanning and placement position these cells on the chip, and routing connects them with metal wires. Each stage can take hours to weeks for complex designs, and engineers tune many parameters along the way, such as clock frequency, cell sizes, placement density, and routing constraints. Together, these parameters determine whether the chip meets its timing requirements. Timing is verified with Static Timing Analysis (STA), the industry-standard method. STA can be run after every stage, but it only uses the information available at that stage: before routing, wire delays are unknown, cell delay models are simplified, and physical design constraints are not yet applied, so early-stage STA gives inaccurate estimates of the final, post-routing timing. The accurate timing is therefore known only after routing, and discovering violations at that point is expensive: engineers must iterate back to earlier stages, discarding days of computational work and delaying time-to-market by weeks or months.

The late discovery of timing problems presents an important machine learning challenge: can we accurately predict late-stage timing quality from early-stage design data? We introduce Sequential Timing Evaluation via Progressive Knowledge Distillation (*STEP-KD*) to address this challenge. STEP-KD is intended as an early prediction tool: right after logic synthesis, engineers obtain a fast estimate of the post-routing TNS and WNS and can flag designs likely to violate timing for parameter adjustments; relaxing the clock period, changing the placement density, or modifying the architecture, before committing to the expensive physical design flow. This enables design space

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.

exploration at synthesis speed rather than routing speed and avoids many late-stage iterations. Our key insight is that the intermediate design stages, specifically post-floorplan and post-placement, bridge the abstraction gap between our starting stage, post-synthesis, and our prediction target, post-routing. Intermediate stages provide progressively refined physical information that serves as "stepping stones" for knowledge transfer. Rather than attempting a single large leap, *STEP-KD* transfers timing knowledge sequentially through adjacent stages. Each distillation step bridges a manageable gap with well-aligned feature spaces, enabling more effective knowledge transfer than direct approaches. This progressive distillation strategy mirrors the natural hierarchy of the IC design flow itself, where each stage incrementally adds physical details—placement provides cell coordinates, clock tree synthesis (CTS) adds clock network information, and routing completes the physical picture.

Our contributions are fourfold: (1)We propose the first progressive knowledge distillation approach for IC timing prediction that explicitly leverages intermediate design stages as sequential teachers, advancing beyond single-step distillation methods. (2) We demonstrate that this stage-sequential approach outperforms direct distillation baselines, validating our hypothesis that intermediate stages provide better representation alignment than attempting to bridge the full abstraction gap in one step. (3) We construct a dataset of about 12k circuit implementations of 922 RTL designs, each represented at four design stages together with the source RTL, which we will release publicly upon acceptance. (4) We provide extensive experimental validation on diverse circuit designs, showing that STEPKD enables accurate post-routing timing prediction from post-synthesis data. Taken jointly, our contributions offer a substantial step towards the goal of providing circuit engineers with instant feedback, allowing them to identify timing violations earlier in the design cycle and avoid costly late-stage iterations.

2 Related Work

2.1 Circuit Timing Prediction

Recent advances in machine learning for electronic design automation (EDA) have demonstrated strong potential for predicting design metrics and guiding decision-making across various stages of the physical design flow (Ren et al., 2022; Ma et al., 2020; Xu et al., 2023; Moravej et al., 2025). In particular, a growing body of work (Fang et al., 2025b; 2023; 2024a; Amik et al., 2025b; Fang et al., 2025a; Wang et al., 2023) explores the use of graph-based and language-based models for predicting computationally intensive metrics such as power, performance, and area (PPA) early in the design flow. For PPA prediction at physical design stages, Barboza et al. (2019) investigates machine learning techniques for post-routing net delay and slew prediction using net features extracted from placement results, but does not directly output global timing metrics. Building on this line of research, Guo et al. (2022) presents a timing engine-inspired GNN framework that accurately predicts post-routing arrival times and slacks. However, both of these methods operate at the post-placement stage and are not readily extendable to earlier design stages.

Several recent studies have attempted to evaluate circuit quality at the early RTL stage. RTL-Timer (Fang et al., 2024b) proposes a Boolean operator graph (BOG) representation of RTL designs and trains a model to predict post-synthesis endpoint arrival times. Amik et al. (2025a) achieves accurate TNS and WNS estimation at the RTL stage using an LLM-augmented GNN framework trained on synthesized RTL datasets. However, these approaches are confined to the logic synthesis stage and do not incorporate layout-aware timing features. To bridge the gap between layout and RTL, Wang et al. (2025) introduces a multi-granularity knowledge distillation framework that transfers physical information from a layout-aware teacher model to an RTL-level student model. NetTag (Fang et al., 2025a) proposes a layout-aware foundational netlist model through self-supervised pretraining and cross-stage alignment via contrastive learning. Despite these advances, by estimating timing at the RTL stage both approaches bypass the physical design process, in which diverse physical information emerges across the layout stages. They also predict timing before technology mapping, without the library-specific cell delays, drive strengths, pin capacitances and setup/hold timing arcs on which path delays fundamentally depend. We provide a detailed comparison of our work with previous related on in Appendix D.

2.2 Privileged Knowledge Distillation in Machine Learning

Our work also relates to the broader literature in machine learning on knowledge distillation between a teacher and a student model, when the teacher has access to more inputs than the student, known as privileged knowledge distillation.

An influential early approach on this topic for classification models is Generalized Distillation (Lopez-Paz et al., 2016), which proposes to train the student to match the gold labels, but adds a regularizer which encourages the student logits to match the teacher logits. Later work has also extended knowledge distillation beyond outputs. For example, Tung & Mori (2019) encourage internal activations of convolutional neural networks to match using a similarity loss regularizer, and Aguilar et al. (2020) encourage attention weights and hidden vector representations of transformer networks to match using Kullback-Leibler and cosine similarity regularizers. Like these works, we also encourage internal representations to match between student and teacher models, although we employ graph neural networks. Most related to ours is Teacher Assistant Knowledge Distillation (Mirzadeh et al., 2020), in which the authors introduce a "teaching assistant" that sits between the teacher and the student: the teacher teaches the TA, and the TA teaches the student. Our framework can be seen as an extension of this idea beyond three networks, although the focus of the authors was to bridge gaps in model capacity, whereas our method focuses on overcoming the information gaps between different stages.

3 Background: Circuit Representation and Timing Analysis

A synchronous digital circuit is represented as a directed graph $G = (V, E)$ whose nodes are circuit elements and whose edges are signal connections. The node set contains (i) sequential elements (registers/flip-flops) or output gates $V_{\text{seq}} \subset V$, which serve as the start and end points of timing paths, and (ii) combinational logic gates, which perform Boolean operations on their inputs. Each edge (u, v) carries a delay $d(u, v)$ comprising the switching delay of the driving gate and the wire delay. The sequential elements are fixed across design stages, whereas the combinational logic may be altered from one stage to the next. STA decomposes the circuit into timing paths, each running from one sequential element through combinational logic to another, and computes the **slack** of every sequential element as the difference between the time its signal is required and the time it actually arrives, $\text{slack}(v) = \text{RAT}(v) - \text{AAT}(v)$ (see Appendix A for details). A positive slack is a timing margin; a negative slack is a **timing violation**, meaning the circuit fails at the target clock period. Because delays and clock timing are refined as the design progresses, slack estimated at an early stage can differ substantially from post-routing slack. We predict the post-routing slack of the sequential elements and summarize circuit timing with the **total negative slack** (TNS) and the **worst negative slack** (WNS),

$$\text{TNS} = \sum_{\substack{v \in V_{\text{seq}} \\ \text{slack}(v) < 0}} \text{slack}(v), \qquad \text{WNS} = \min_{v \in V_{\text{seq}}} \text{slack}(v). \tag{1}$$

TNS quantifies the aggregate severity of all timing violations in the circuit, and WNS the single most critical one: a circuit can have a small WNS but a large TNS (many moderately failing paths), or a large WNS but a small TNS (one severely failing path).

4 Method: STEP-KD

We present STEP-KD, our method for predicting post-routing timing from post-synthesis circuit data. STEP-KD uses the same graph neural network architecture at every design stage (Section 4.1) and trains one model per stage, one at a time, from the latest stage to the earliest: each model is distilled from all later-stage models through a cascade of learned projections (Section 4.3). Once training is complete, only the post-synthesis model is kept for inference. Figure 1 summarizes the training procedure and the model architecture.

4.1 Features and Encoding

Our models take as input graph representations of circuits at four stages: post-synthesis (PS), post-floorplan (PF), post-placement (PP), and post-routing (PR). Each node carries a stage-dependent

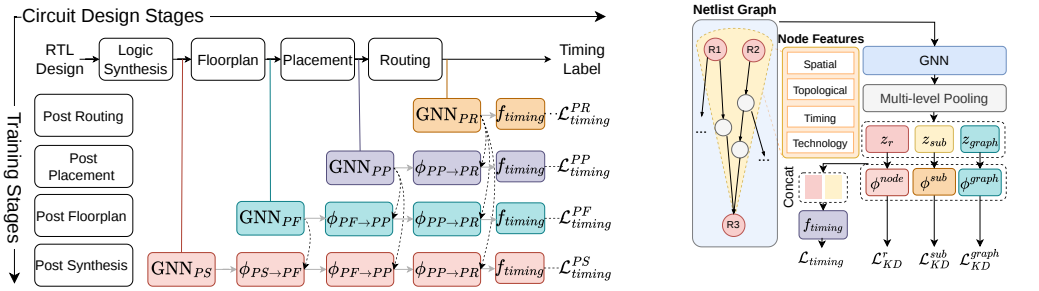


Figure 1: **Training flow (left)**: Models are trained one at a time down the design ladder, in the order PR, PP, PF, PS. Each model is trained to predict post-routing slack and, except for PR, is regularized by knowledge distillation (KD) losses that align its representations, passed through its own cascade of learned projections, with those of the frozen models at all later stages, which have access to physical information not yet available at its own stage. Only the post-synthesis model is used at inference. **Model architecture (right)**: The architecture of our machine learning model. It takes as input a graphical representation of a circuit at the post-synthesis stage (netlist graph) and outputs the slack for every sequential element in the circuit. From these predictions, circuit-wide metrics such as the total negative slack (TNS) and worst negative slack (WNS) can be computed.

feature vector $\mathbf{x}_v \in \mathbb{R}^{d_{\text{in}}}$ that concatenates **spatial** features (physical coordinates (x_v, y_v), available from post-placement onward and set to $(0, 0)$ at the PS and PF stages), **timing** features (the current-stage slack from STA), **topological** features (one-hot cell type, depth-per-input, depth-per-output, and fan-in/fan-out counts), and **technology** features (drive strength, pin counts, library parameters, and the design-level placement density, core utilization and clock period); see Appendix B.3 for details. At each stage, a Graph Convolutional Network (GCN) with L layers of neighborhood aggregation encodes these features into node embeddings $\mathbf{Z} = \{\mathbf{z}_v \in \mathbb{R}^{d_{\text{emb}}} : v \in V\}$, which capture both local neighborhood structure and global circuit properties.

4.2 Multi-Granularity Representations and Projections

Multi-granularity pooling. From the node embeddings we extract representations at three granularities: the node-level embedding $\mathbf{z}_r$ of each sequential element $r \in V_{\text{seq}}$; a subgraph-level embedding $\mathbf{z}_r^{\text{sub}} = \frac{1}{|V_{\text{sub}}(r)|} \sum_{v \in V_{\text{sub}}(r)} \mathbf{z}_v$, obtained by mean pooling over the fan-in cone $G_{\text{sub}}(r) = (V_{\text{sub}}(r), E_{\text{sub}}(r))$ of r, i.e., all nodes and edges between r and its source sequential elements; and a graph-level embedding $\mathbf{z}^{\text{graph}} = \frac{1}{|V|} \sum_{v \in V} \mathbf{z}_v$. At stage s this gives $\mathbf{Z}^s = \{\mathbf{z}_r^s, \mathbf{z}_r^{s,\text{sub}}, \mathbf{z}^{s,\text{graph}} : r \in V_{\text{seq}}\}$, capturing individual sequential elements, their local combinational logic, and global circuit properties. The node-level representation also carries a learned embedding of the current-stage STA slack (Appendix C).

Cascaded projections. We index the stages in design order, $s_0 = \text{PS}$, $s_1 = \text{PF}$, $s_2 = \text{PP}$, $s_3 = \text{PR}$. A projection $\Phi_{s_k \to s_{k+1}}$ maps representations of one stage into the representation space of the next. It consists of three three-layer MLPs $\phi^{\text{node}}, \phi^{\text{sub}}, \phi^{\text{graph}} : \mathbb{R}^{d_{\text{emb}}} \to \mathbb{R}^{d_{\text{emb}}}$, one per granularity, each with a residual connection around every linear layer. The model at stage s_k learns its own cascade of projections, one per design step, whose composition $\Phi_{s_k \to s_j} = \Phi_{s_{j-1} \to s_j} \circ \cdots \circ \Phi_{s_k \to s_{k+1}}$ maps its representations into the space of any later stage s_j. Each single projection $\Phi_{s_k \to s_{k+1}}$ thus bridges only one design step. For example, the PF model reaches the PR space through the PP space with the map $\Phi_{s_1 \to s_3} = \Phi_{s_2 \to s_3} \circ \Phi_{s_1 \to s_2}$. For a (possibly composed) projection Φ with granularity components $\phi^{\text{node}}, \phi^{\text{sub}}, \phi^{\text{graph}}$, the alignment with target representations $\mathbf{Z}'$ is a weighted sum over granularities,

$$\|\Phi(\mathbf{Z}) - \mathbf{Z}'\|^2 = \frac{\beta_1}{m} \sum_{r \in V_{\text{seq}}} \|\phi^{\text{node}}(\mathbf{z}_r) - \mathbf{z}'_r\|_2^2 + \frac{\beta_2}{m} \sum_{r \in V_{\text{seq}}} \|\phi^{\text{sub}}(\mathbf{z}_r^{\text{sub}}) - \mathbf{z}_r'^{\text{sub}}\|_2^2 + \beta_3 \|\phi^{\text{graph}}(\mathbf{z}^{\text{graph}}) - \mathbf{z}'^{\text{graph}}\|_2^2, \tag{2}$$

where $m = |V_{\text{seq}}|$ and $\beta_1, \beta_2, \beta_3$ are hyperparameters weighting the importance of each granularity. This multi-granularity alignment ensures that the student learns to match each teacher's representations at multiple levels of abstraction – from individual sequential elements to local combinational logic to global circuit properties.

Timing predictor. The timing predictor f^s_{timing} of the stage-s model predicts the post-routing slack of each sequential element $r \in V_{\text{seq}}$ from its node- and subgraph-level representations after projection to the PR space. Writing $\tilde{\mathbf{z}}^s_r = \phi^{\text{node}}_{s\to\text{PR}}(\mathbf{z}^s_r)$ and $\tilde{\mathbf{z}}^{s,\text{sub}}_r = \phi^{\text{sub}}_{s\to\text{PR}}(\mathbf{z}^{s,\text{sub}}_r)$, where $\phi^{\text{node}}_{s\to\text{PR}}$ and $\phi^{\text{sub}}_{s\to\text{PR}}$ are the node- and subgraph-level components of $\Phi_{s\to\text{PR}}$ (set to identity for $s = \text{PR}$), the predictor computes

$$\widehat{\text{slack}}_{\text{PR}}(r) = f^s_{\text{timing}}\big([\tilde{\mathbf{z}}^s_r;\ \tilde{\mathbf{z}}^{s,\text{sub}}_r]\big), \tag{3}$$

where $[\cdot;\cdot]$ denotes concatenation and $f^s_{\text{timing}} : \mathbb{R}^{2d_{\text{emb}}} \to \mathbb{R}$ is an MLP. The graph-level representation enters training only through the distillation loss.

4.3 Training

STEP-KD trains one model per design stage and climbs down the design ladder: PR first, then PP, PF and finally PS. All four models predict the same post-routing slack labels. Each model is trained only after every later-stage model has been trained and frozen, and it is distilled from all of them; the PR model, which has no later stage, is trained with supervision alone. The PP model thus learns from PR, the PF model from PP and PR, and the PS model from PF, PP and PR (Figure 1, left). Each teacher learns to predict post-routing timing metrics using data from its respective stage.

The model at stage s_k predicts post-routing slack from its PR-space representations and aligns its projected representations with the encoder outputs of every frozen later-stage model,

$$\begin{aligned}\mathcal{L}^{s_k} = \frac{\gamma_{\text{timing}}}{|V_{\text{seq}}|} \sum_{r\in V_{\text{seq}}} \Big|\widehat{\text{slack}}_{\text{PR}}(r) - \text{slack}_{\text{PR}}(r)\Big| \\ + \gamma_{\text{KD}} \sum_{j=k+1}^{3} \big\|\Phi_{s_k\to s_j}(\mathbf{Z}^{s_k}) - \mathbf{Z}^{s_j}\big\|^2.\end{aligned} \tag{4}$$

The KD sum is empty when training the PR model. During the PP model training, the KD term aligns the representations of the PP model with the trained PR model. During the PF model training, it aligns the representations of the model with the PP and PR models. Finally, during the PS model training, it aligns the student with the PF, PP and PR models after one, two and three projection steps. Because the PF and PP teachers were themselves distilled from later stages, their representations are already aligned with those of the stages above them, so knowledge from routing reaches the student one rung at a time. Table 2 compares this against independently trained teachers.

4.4 Inference

At inference, only the PS model is used: its representations pass through its projection cascade (PS $\to$ PF $\to$ PP $\to$ PR) to its timing predictor $f^{\text{PS}}_{\text{timing}}$. The timing metrics can then be calculated directly from the node-level timing predictions for each circuit using Equation (1), with the predicted $\widehat{\text{slack}}$ values from the post-synthesis graph. This gives an early estimate of the PR timing information, saving expensive iterations in the design flow. [1]

5 Experiments

5.1 Dataset Generation

To train and evaluate STEP-KD, we require circuit representations at all four design stages (post-synthesis, post-floorplan, post-placement, post-routing) for the same designs. We construct this dataset using open-source RTL designs and an automated EDA flow.

[1] The sequential nodes are in one-to-one correspondence across design stages, so the sums range over the same set of nodes.

Source RTL Designs. We collect 24,200 Verilog designs from two sources: 18,400 examples from Pei et al. (2024) and 5,800 from Thakur et al. (2022). These designs originate from open-source GitHub repositories and hardware design textbooks, covering arithmetic units, control logic, memory controllers, and communication interfaces.

Multi-Stage Circuit Generation. We run each design through logic synthesis and the complete physical design flow using the open-source OpenROAD platform (Ajayi et al., 2019) with the ASAP7 7nm predictive PDK. To obtain diverse implementations, we sweep the placement density over $\{0.7, 0.75, 0.8\}$ and the core utilization over $\{40, 45, 50, 55, 60\}\%$, giving 15 configurations per design; the clock period is set to OpenROAD's default for each design and used as an input feature. Of the 363K attempted runs, approximately 6% complete all four stages with valid timing analysis, the rest failing because of, e.g., routing congestion or placement density conflicts. This yields 22k circuit implementations. Using 72 Core Intel(R) Xeon(R) Gold 6140 CPUs, each of the 15 runs took about one week to complete. After removing circuits with fewer than 150 nodes at the post-synthesis stage as trivial, we obtain 922 designs and 12.2k circuit implementations, each represented at all four stages; statistics are given in Appendix B. We will release the full dataset, including the source RTL and the graph representations at all four stages, upon acceptance of this paper.

Dataset Split and Labels. We split the source circuits into a 70/10/20 train/validation/test split. The three sets consist of 8.6k, 1.2k and 2.2k circuits, respectively (with 650, 94, and 178 unique RTL designs). All circuits derived from the same RTL design belong to the same split, so models are evaluated on unseen designs rather than on parameter variations of seen ones. The labels at every stage are the post-routing slack values from OpenROAD's final static timing analysis report.

Additional Benchmark. We further evaluate all methods on circuits from the OpenLLM-RTL benchmark (Liu et al., 2025) as a complementary test set. Specifically, we use its RTLLM 2.0 designs: 50 diverse, simple, hand-crafted RTL circuits spanning arithmetic, memory, control, and processor-oriented functionality, in contrast to our primary test set, which consists of larger and more structurally complex open-source RTL designs. This diversity in circuit functionality allows us to assess whether the proposed approach generalizes across different types of RTL designs. None of these circuits are used during training. We pass them through the same flow with the same parameters, obtaining 189 circuits of 38 unique designs.

5.2 Training and Evaluation Setup

Model Architecture. We implement our graph neural network encoder using PyTorch Geometric. The GCN consists of 3 message-passing layers with a hidden dimension of 128, ReLU activation functions, and layer normalization. For the projection networks $\Phi_{s_1 \to s_2}$, each three-layer MLP uses a hidden dimension of 128, ReLU activations, and a residual connection around each linear layer, which we find lowers the alignment loss and improves STEP-KD performance. The timing predictor (Section 4.2) is implemented as a 2-layer MLP.

Training Configuration. We train all models using the Adam optimizer with an initial learning rate of 10^{-4}. We employ a ReduceLROnPlateau scheduler with patience of 20 epochs and a decay factor of 0.5. We train all models until convergence and keep the checkpoint with the lowest validation loss. For STEP-KD, the models are trained one at a time in the order PR, PP, PF, PS (Section 4.3), and each model is frozen before the next one is trained.

Evaluation Protocol. Each of the 178 test designs has several implementations (different placement density and core utilization). We conduct 20 evaluation runs, each sampling one implementation per design, and report the mean and standard deviation over these runs of the median absolute error (MedAE, in the units of the slack labels), WMAPE (Weighted Mean Absolute Percentage Error), and RSE (Relative Squared Error); Table 5 in the appendix additionally reports R^2 and the mean absolute error (MAE). The three main metrics differ deliberately in their sensitivity to outliers. MedAE ignores them: an anomaly does not shift the median unless it affects more than half of the circuits. WMAPE divides the total absolute error by the total magnitude of the labels, so it is robust to large relative errors on circuits with small slack but sensitive to large errors on circuits with large slack. RSE squares the errors, so a single large mistake can dominate it.

Table 1: Performance of methods taking post-synthesis inputs and predicting post-routing timing on the held-out test set (top) and the OpenLLM-RTL benchmark (bottom). Direct KD distills from a single teacher (stage in parentheses). STEP-KD w/o sequential KD aligns the student to independently trained PF, PP and PR teachers; STEP-KD uses teachers that were themselves distilled along the design ladder (PR→PP→PF). MedAE (median absolute error) is in the units of the slack labels. Best per dataset in bold.

	TNS			WNS		
Method	MedAE ↓	WMAPE (%) ↓	RSE (%) ↓	MedAE ↓	WMAPE (%) ↓	RSE (%) ↓
Held-out test set						
STA	512.61 ± 20.42	74.84 ± 0.60	145.49 ± 3.11	23.69 ± 0.56	35.59 ± 1.53	46.52 ± 14.72
No KD	740.05 ± 52.44	38.75 ± 2.20	16.83 ± 4.06	25.82 ± 1.73	44.49 ± 2.23	62.14 ± 14.24
Direct KD (PR teacher)	399.07 ± 27.79	24.05 ± 1.51	11.24 ± 3.13	15.45 ± 0.99	28.64 ± 2.30	45.16 ± 18.84
Direct KD (PP teacher)	396.84 ± 31.08	24.49 ± 1.84	10.95 ± 3.71	16.56 ± 0.83	29.14 ± 2.12	45.33 ± 19.40
Direct KD (PF teacher)	421.20 ± 30.84	23.65 ± 1.54	9.79 ± 3.37	18.23 ± 0.97	29.29 ± 2.31	43.03 ± 20.25
STEP-KD w/o sequential KD	403.09 ± 27.43	30.18 ± 1.79	13.30 ± 3.36	16.35 ± 0.59	26.91 ± 1.88	44.96 ± 17.65
STEP-KD	$\mathbf{302.27 \pm 18.37}$	$\mathbf{19.78 \pm 1.54}$	$\mathbf{7.85 \pm 2.95}$	$\mathbf{11.09 \pm 0.41}$	$\mathbf{21.39 \pm 1.97}$	$\mathbf{41.09 \pm 17.81}$
OpenLLM-RTL benchmark						
STA	$\mathbf{50.76 \pm 0.83}$	41.84 ± 0.43	43.42 ± 0.50	$\mathbf{13.24 \pm 0.25}$	$\mathbf{33.71 \pm 0.39}$	16.88 ± 0.68
No KD	188.01 ± 17.06	79.37 ± 4.03	52.07 ± 9.63	37.05 ± 3.23	99.34 ± 3.75	158.13 ± 7.75
Direct KD (PR teacher)	73.55 ± 6.72	43.78 ± 2.59	25.08 ± 4.51	21.18 ± 1.70	54.62 ± 2.20	43.40 ± 2.22
Direct KD (PP teacher)	72.99 ± 5.98	49.84 ± 3.14	42.04 ± 9.76	16.51 ± 1.32	45.25 ± 2.21	32.44 ± 2.89
Direct KD (PF teacher)	101.47 ± 7.10	38.23 ± 3.00	14.86 ± 4.50	23.38 ± 1.73	50.30 ± 1.67	31.22 ± 1.86
STEP-KD w/o sequential KD	76.74 ± 5.08	32.41 ± 2.67	11.93 ± 4.43	14.10 ± 0.71	35.58 ± 0.61	20.08 ± 0.71
STEP-KD	62.43 ± 4.14	$\mathbf{27.43 \pm 1.41}$	$\mathbf{6.49 \pm 1.23}$	15.98 ± 0.69	36.55 ± 0.72	$\mathbf{16.52 \pm 0.64}$

5.3 Baselines and Model Variants

We compare STEP-KD against three baselines and one ablated variant.

Post-Synthesis STA. OpenROAD's static timing analysis at the post-synthesis stage, the standard industry practice for early-stage timing estimation.

No KD. A GNN with the same encoder, pooling and timing predictor as STEP-KD, trained on post-synthesis graphs with the timing loss alone. It tests whether direct supervised prediction can bridge the synthesis-to-routing gap without intermediate-stage information.

Direct KD. Single-step distillation from one frozen teacher at the PR, PP or PF stage to the post-synthesis student through a single multi-granularity projection, similar to RTLDistil (Wang et al., 2025). Each teacher is trained independently without a KD loss. With the PR teacher it attempts to bridge the full synthesis-to-routing gap in one step, isolating the benefit of progressive over direct distillation.

STEP-KD w/o sequential KD. Identical to STEP-KD for the PS student, but its PF and PP teachers are trained independently rather than distilled from later stages, isolating the contribution of distilling the teachers themselves.

6 Results

Main results. Table 1 summarizes the comparison. On the held-out test set, STEP-KD achieves the lowest TNS and WNS error on all three metrics. On both datasets it outperforms No KD and every Direct KD variant on every metric, showing that distilling through the intermediate stages is more effective than distilling from any single stage. On OpenLLM-RTL, STEP-KD has the lowest TNS WMAPE and RSE, and on WNS it is on par with post-synthesis STA, which is considerably more accurate on this benchmark than on the held-out set. Without distillation, the learned model is less accurate than STA on WNS on both datasets, so distillation is essential for this metric.

Robustness to outliers. Comparing the three metrics, which differ in their sensitivity to outliers (Section 5.2), together with the scatter plots in Figure 2 shows how the errors are distributed. On the held-out set, the WNS RSE of every method is high and varies strongly across evaluation runs, even though STEP-KD's WNS MedAE and WMAPE are low. The scatter plots explain this: a handful of circuits have a post-routing WNS far larger than predicted from the post-synthesis netlist (upper

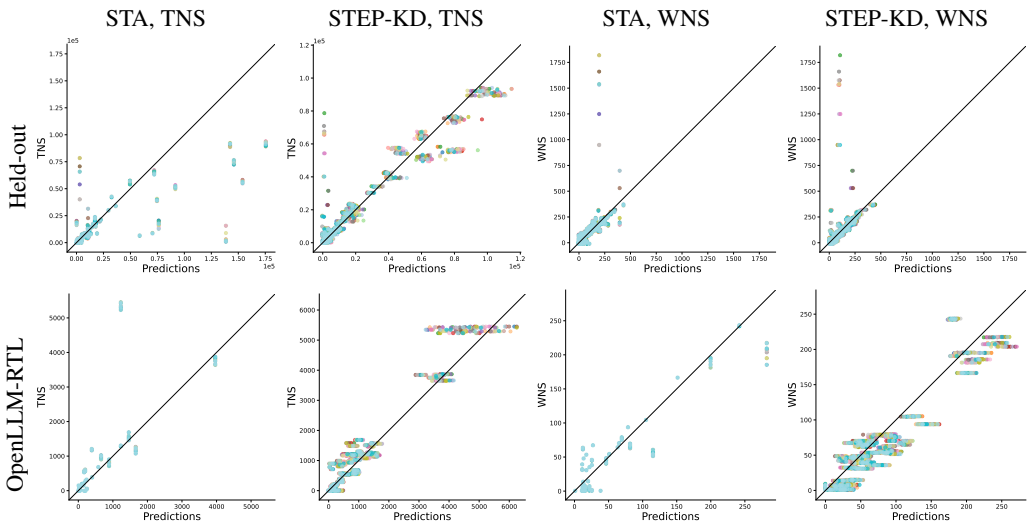


Figure 2: Predicted versus post-routing TNS and WNS (magnitudes) for post-synthesis STA and STEP-KD on the held-out test set (top) and the OpenLLM-RTL benchmark (bottom). The diagonal marks perfect prediction. Each color is one of the 20 evaluation runs, each of which samples one implementation per design.

Table 2: STA, teacher models and the effect of sequential knowledge distillation (held-out test set). At each intermediate stage we compare STA, a model trained without distillation from later stages, and a model distilled along the design ladder. At PS both STEP-KD variants align to PF, PP and PR teachers one at a time and differ only in whether those teachers were themselves distilled; post-synthesis STA is reported in Table 1. MedAE (median absolute error) is in the units of the slack labels. Best within each stage in bold. †The PR teacher receives post-routing slack as an input feature, so its near-perfect score is expected and shown only for completeness.

		TNS			WNS		
Stage	Predictor	MedAE ↓	WMAPE (%) ↓	RSE (%) ↓	MedAE ↓	WMAPE (%) ↓	RSE (%) ↓
PR	STA	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.00 ± 0.00
	Teacher†	32.61 ± 2.88	1.74 ± 0.15	0.06 ± 0.03	1.12 ± 0.08	1.88 ± 0.22	0.23 ± 0.21
PP	STA	378.28 ± 16.03	63.57 ± 0.91	105.06 ± 3.82	18.80 ± 0.61	29.71 ± 1.06	22.64 ± 5.65
	Independent (no KD)	355.77 ± 24.70	19.61 ± 1.52	6.01 ± 2.13	$\mathbf{12.11 \pm 0.73}$	$\mathbf{21.22 \pm 1.53}$	20.09 ± 9.96
	KD from PR	$\mathbf{306.37 \pm 19.85}$	$\mathbf{17.29 \pm 1.02}$	$\mathbf{4.56 \pm 1.19}$	14.46 ± 0.60	22.78 ± 1.08	$\mathbf{17.06 \pm 5.26}$
PF	STA	424.67 ± 16.87	57.39 ± 0.58	85.85 ± 2.42	18.36 ± 0.70	30.29 ± 0.91	24.41 ± 5.63
	Independent (no KD)	341.25 ± 28.51	20.66 ± 1.72	6.48 ± 2.13	15.57 ± 0.78	23.46 ± 1.47	$\mathbf{20.74 \pm 9.82}$
	KD from PP (PR→PP)	$\mathbf{273.94 \pm 19.38}$	$\mathbf{17.43 \pm 1.62}$	$\mathbf{5.38 \pm 2.19}$	$\mathbf{12.48 \pm 0.59}$	$\mathbf{20.58 \pm 1.90}$	20.80 ± 9.64
PS	STEP-KD w/o sequential KD	403.09 ± 27.43	30.18 ± 1.79	13.30 ± 3.36	16.35 ± 0.59	26.91 ± 1.88	44.96 ± 17.65
	STEP-KD	$\mathbf{302.27 \pm 18.37}$	$\mathbf{19.78 \pm 1.54}$	$\mathbf{7.85 \pm 2.95}$	$\mathbf{11.09 \pm 0.41}$	$\mathbf{21.39 \pm 1.97}$	$\mathbf{41.09 \pm 17.81}$

left of the WNS panels), and neither STA nor STEP-KD anticipates this jump; similar outliers appear for TNS. WNS is intrinsically harder to predict than TNS: as the minimum slack over all sequential elements, it is set by a single critical path, so any error on that one path carries over entirely to WNS, whereas TNS aggregates the slack of many endpoints. The circuits of OpenLLM-RTL are much simpler, so their timing changes little between post-synthesis and post-routing, and post-synthesis STA is a reasonable estimate: it has the lowest TNS MedAE and is on par with STEP-KD on WNS. Nonetheless, STEP-KD has a far lower TNS WMAPE and RSE, because STA grossly underestimates a few circuits with large TNS (e.g., near 5,000) that STEP-KD predicts much more closely. Appendix E also shows that STEP-KD, unlike STA, adapts its predictions to the physical design parameters of each implementation.

Teacher models and sequential distillation. Table 2 reports the models at the intermediate stages. On the held-out set, the teachers are more accurate than STA at their own stage on every metric, and distilling them along the ladder further improves their TNS predictions, with a mixed effect on WNS. The effect is largest at the student: aligning the PS model to ladder-distilled teachers instead of independently trained ones improves all six metrics. Without sequential distillation, the

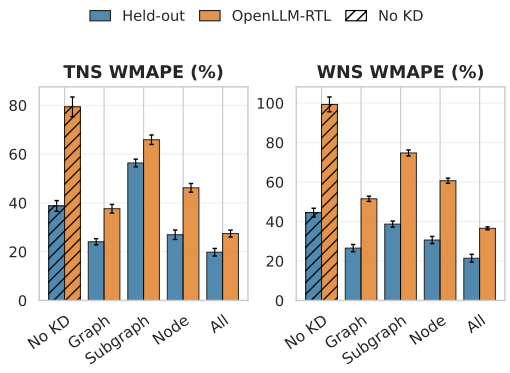


Figure 3: Impact of multi-granularity distillation on WMAPE (lower is better). Each single-granularity variant distills from the same teachers as STEP-KD (*All*) but aligns only the listed granularity; hatched bars show No KD.

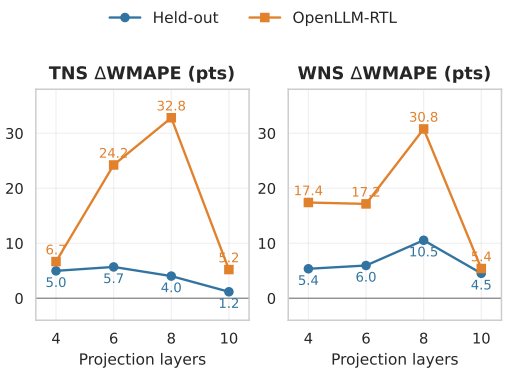


Figure 4: WMAPE of Direct KD (PR teacher) with 4 to 10 projection layers minus the WMAPE of STEP-KD, in percentage points. Positive values mean STEP-KD has lower error.

PS student has a higher TNS WMAPE and RSE than every Direct KD variant, so the benefit of the cascade largely comes from passing knowledge down the ladder rather than from aligning to more teachers.

Impact of multi-granular distillation. To measure the contribution of each granularity, we train PS students that distill from the same teachers as STEP-KD but align only the graph-, subgraph- or node-level representations. Aligning all three granularities gives the lowest error on both datasets (Figure 3), and on every metric (Tables 6 and 9). Graph-level alignment is the most effective single granularity, followed by node-level alignment. Subgraph-level alignment alone is the weakest and on held-out TNS is even worse than No KD, yet it helps in combination with the other two, so the granularities are complementary.

Effect of projection network depth. STEP-KD's PS model composes three 3-layer projections. To test whether its advantage comes from this added capacity rather than from the design ladder, we vary the depth of the single projection of Direct KD (PR teacher) from 4 to 10 layers. STEP-KD is more accurate than Direct KD at every depth, on both datasets and every metric (Figure 4; Tables 7 and 10). Deeper projections do not reliably narrow the gap and can be unstable, and even the 10-layer projection, deeper than STEP-KD's 9-layer cascade, still trails it. The advantage of STEP-KD therefore does not come from depth alone.

7 Conclusion

We presented STEP-KD, a novel approach for predicting post-routing timing from post-synthesis circuit data. Our key insight is that leveraging intermediate design stages, post-floorplan and post-placement, as sequential teachers enables more effective knowledge transfer than direct distillation. STEP-KD can potentially enable engineers to identify timing violations earlier in the design cycle, helping to cut costs and avoiding expensive late-stage redesigns.

AI use statement

All code implementing the methods and constructing the dataset was written by the authors, without AI-generated code. The main paper body was written by the authors; Large Language Models were used to polish and revise the text and to help produce figures and tables from our experimental results. The authors reviewed all AI-assisted text, figures and tables and take full responsibility for the content of the paper.

Reproducibility statement

Section 5.1 describes the construction of the dataset, Section 5.2 the model architecture, training configuration and evaluation protocol, and Appendix B.3 the input features. The full dataset, including the source RTL and the graph representations at all four design stages, will be released upon acceptance.

References

Gustavo Aguilar, Yuan Ling, Yu Zhang, Benjamin Yao, Xing Fan, and Chenlei Guo. Knowledge distillation from internal representations. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 7350–7357, 2020.

T Ajayi, D Blaauw, TB Chan, CK Cheng, VA Chhabria, DK Choo, M Coltella, S Dobre, R Dreslinski, M Fogaça, et al. Openroad: Toward a self-driving, open-source digital layout implementation tool chain. In *GOMACTECH*, 2019.

Fahad Rahman Amik, Yousef Safari, Zhanguang Zhang, and Boris Vaisband. *Graph-Based Timing Prediction at Early-Stage RTL Using Large Language Model*. 2025a.

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 ACM/IEEE Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 393–400, 2025b.

Erick Carvajal Barboza, Nishchal Shukla, Yiran Chen, and Jiang Hu. Machine learning-based pre-routing timing prediction with reduced pessimism. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019.

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 *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–9, 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 (DAC)*, 2024a.

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, 2024b.

Wenji Fang, Wenkai Li, Shang Liu, Yao Lu, Hongce Zhang, and Zhiyao Xie. Nettag: A multimodal rtl -and- layout -aligned netlist foundation model via text - attributed graph. In *ACM/IEEE Design Automation Conference (DAC)*, 2025a.

Wenji Fang, Shang Liu, Jing Wang, and Zhiyao Xie. Circuitfusion: Multimodal circuit representation learning for agile chip design. In *The Thirteenth International Conference on Learning Representations (ICLR)*, 2025b.

Zizheng Guo, Mingjie Liu, Jiaqi Gu, Shuhan Zhang, David Z. Pan, and Yibo Lin. A timing engine inspired graph neural network model for pre-routing slack prediction. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022.

Shang Liu, Yao Lu, Wenji Fang, Mengming Li, and Zhiyao Xie. Openllm-rtl: Open dataset and benchmark for llm-aided design rtl generation. In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '24, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400710773. doi: 10.1145/3676536.3697118. URL https://doi.org/10.1145/3676536.3697118.

David Lopez-Paz, Léon Bottou, Bernhard Schölkopf, and Vladimir Vapnik. Unifying distillation and privileged information. In Yoshua Bengio and Yann LeCun (eds.), *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

Yuzhe Ma, Zhuolun He, Wei Li, Lu Zhang, and Bei Yu. Understanding graphs in eda: From shallow to deep learning. In *Proceedings of the 2020 international symposium on physical design*, pp. 119–126, 2020.

Seyed Iman Mirzadeh, Mehrdad Farajtabar, Ang Li, Nir Levine, Akihiro Matsukawa, and Hassan Ghasemzadeh. Improved knowledge distillation via teacher assistant. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 5191–5198, 2020.

Reza Moravej, Saurabh Bodhe, Zhanguang Zhang, Didier Chételat, Dimitrios Tsaras, Yingxue Zhang, Hui-Ling Zhen, Jianye Hao, and Mingxuan Yuan. The graph's apprentice: Teaching an llm low-level knowledge for circuit quality estimation. In James Kwok (ed.), *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence, IJCAI-25*, pp. 9296–9304. International Joint Conferences on Artificial Intelligence Organization, 8 2025. doi: 10.24963/ijcai.2025/1033. URL https://doi.org/10.24963/ijcai.2025/1033. AI4Tech: AI Enabling Technologies.

Zehua Pei, Huiling Zhen, Mingxuan Yuan, Yu Huang, and Bei Yu. BetterV: Controlled verilog generation with discriminative guidance. In *International Conference on Machine Learning (ICLR)*, 2024.

Haoxing Ren, Siddhartha Nath, Yanqing Zhang, Hao Chen, and Mingjie Liu. Why are graph neural networks effective for eda problems? In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pp. 1–8, 2022.

Shailja Thakur, Baleegh Ahmad, Zhenxing Fan, Hammond A. Pearce, Benjamin Tan, Ramesh Karri, Brendan Dolan-Gavitt, and Siddharth Garg. Benchmarking large language models for automated verilog rtl code generation. *Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2022.

Fred Tung and Greg Mori. Similarity-preserving knowledge distillation. In *International Conference on Computer Vision (ICCV)*, pp. 1365–1374, 2019.

Mingjun Wang, Yihan Wen, Bin Sun, Jianan Mu, Juan Li, Xiaoyi Wang, Jing Justin Ye, Bei Yu, and Huawei Li. Bridging layout and RTL: Knowledge distillation based timing prediction. In *Forty-second International Conference on Machine Learning*, 2025.

Ziyi Wang, Siting Liu, Yuan Pu, Song Chen, Tsung-Yi Ho, and Bei Yu. Restructure-tolerant timing prediction via multimodal fusion. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2023.

Ceyu Xu, Pragya Sharma, Tianshu Wang, and Lisa Wu Wills. Fast, robust and transferable prediction for hardware logic synthesis. MICRO '23, pp. 167–179, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400703294.

A CIRCUIT TIMING DEFINITIONS

This appendix details the timing concepts summarized in Section 3.

Timing paths. A timing path $P = (v_0, v_1, \ldots, v_k)$ is a sequence of nodes connected by directed edges, starting at a register or primary input $v_0 \in V_{\text{seq}}$ and ending at a register or primary output $v_k \in V_{\text{seq}}$. The intermediate nodes $v_1, \ldots, v_{k-1}$ represent combinational logic. The **path delay** is the cumulative delay along the path,

$$\text{delay}(P) = \sum_{i=0}^{k-1} d(v_i, v_{i+1}), \tag{5}$$

where each edge delay $d(u, v)$ includes the gate delay (the intrinsic switching time of the driving gate) and the interconnect delay (wire resistance-capacitance effects). In a synchronous circuit operating with clock period T_{clk}, signals must propagate through combinational logic and arrive at the destination register before the next clock edge; this is the fundamental timing requirement the circuit must satisfy. STA decomposes a design into timing paths and checks whether each signal meets this requirement, using the delays available at the current design stage. The accuracy of the resulting arrival times, and hence of the slack, depends on the accuracy of the delay values and clock timing information, which improve progressively through the design flow.

Slack. The slack at a node v, $\text{slack}(v) = \text{RAT}(v) - \text{AAT}(v)$, is the difference between its required arrival time (RAT) and actual arrival time (AAT), defined below. A positive slack indicates that the signal arrives earlier than required: the path has a timing margin and could tolerate additional delay. A negative slack means the signal arrives later than required, a timing violation: the circuit will fail to operate correctly at the target clock frequency.

Actual Arrival Time (AAT)

The **actual arrival time** (AAT) at a node v, denoted $\text{AAT}(v)$, represents the time at which a signal actually arrives at that node relative to the clock edge that launches it. For a sequential element node, AAT is initialized to the clock arrival time plus the clock-to-q delay (the time for the sequential element output to become valid after the clock edge):

$$\text{AAT}(v) = t_{\text{clk}}(v) + t_{\text{cq}}(v), \quad \text{for } v \in V_{\text{seq}} \tag{6}$$

For a combinational node $v \in V_{\text{comb}}$, the AAT is computed by considering all possible input paths and taking the maximum (latest) arrival:

$$\text{AAT}(v) = \max_{u \in \text{fanin}(v)} \{\text{AAT}(u) + d(u, v)\} \tag{7}$$

where $\text{fanin}(v) = \{u \in V : (u, v) \in E\}$ is the set of nodes that drive v. This max operation reflects the critical constraint that a combinational gate cannot produce a valid output until all its inputs have arrived and stabilized. The AAT propagates forward through the circuit graph from sequential elements through combinational logic to downstream sequential elements.

Required Arrival Time (RAT)

The **required arrival time** (RAT) at a node v, denoted $\text{RAT}(v)$, represents the latest time at which a signal must arrive at that node to meet timing constraints. This is computed through backward propagation from sequential elements. For a destination sequential element node $v \in V_{\text{seq}}$, the RAT is determined by the clock period and setup time requirement:

$$\text{RAT}(v) = t_{\text{clk}}(v) + T_{\text{clk}} - t_{\text{setup}}(v) \tag{8}$$

where $t_{\text{clk}}(v)$ is the clock arrival time at register v, T_{clk} is the clock period, and $t_{\text{setup}}(v)$ is the setup time (the minimum time before the clock edge that data must be stable). For a combinational node $v \in V_{\text{comb}}$, the RAT is computed by considering all fanout paths and taking the minimum (earliest) required time:

$$\text{RAT}(v) = \min_{w \in \text{fanout}(v)} \{\text{RAT}(w) - d(v, w)\} \tag{9}$$

where $\text{fanout}(v) = \{w \in V : (v, w) \in E\}$ is the set of nodes driven by v. The RAT propagates backward from registers through combinational logic to upstream registers.

B Dataset Details

B.1 Dataset Statistics

We plot the histogram of label and graph size distribution across the dataset in Figure 6. The TNS values fall between 0 and 538124, and the WNS values fall in the range of 0 to 2155. About 12 percent of circuits in the dataset have WNS/TNS values equal to zero.

Figure 5 depicts the correlation between timing labels from different stages. The largest change in labels occurs after floorplanning, while placement does not change the STA produced labels significantly.

B.2 OpenLLM-RTL Benchmark

From the OpenLLM-RTL benchmark, we exclude five circuits whose post-routing TNS and WNS are very large and identical to their post-synthesis values, so that post-synthesis STA is exact on them. Because of their magnitude, these circuits would dominate the WMAPE and RSE of every method and drive all results to near-perfect scores, masking the differences between methods.

B.3 Data Feature Details

- **Spatial Features:** These are the node coordinates, which become available after placement. We extract the coordinates for the post-placement and post-routing circuits, and normalize them per circuit by the maximum (x,y) coordinate in the circuit (so the values fall between zero and one). We assign coordinates (0,0) to all nodes in the post-synthesis and post-floorplan stages.
- **Timing Features:** The timing feature is a single scalar, which specifies the slack value estimated by the current stage STA.
- **Topological Features:** We extract 61 topological and functional features from the technology library reports. These features include *area, fan in, fan out, SI direction, SN direction, capacitance, etc.*

C Input Encoding and Slack Bias

Input encoding. Each node has four feature vectors: the spatial, timing and topological features described in Appendix B.3, and the technology features (Section 4.1). Each vector is passed through a separate linear layer, and the four outputs are concatenated to form the input feature $\mathbf{x}_v \in \mathbb{R}^{d_{\text{in}}}$ of the node.

Slack bias. The GNN encoder outputs a representation $\mathbf{z}_v$ for every node. Before the pooled node-, subgraph- and graph-level representations are passed to the projection cascade and the timing

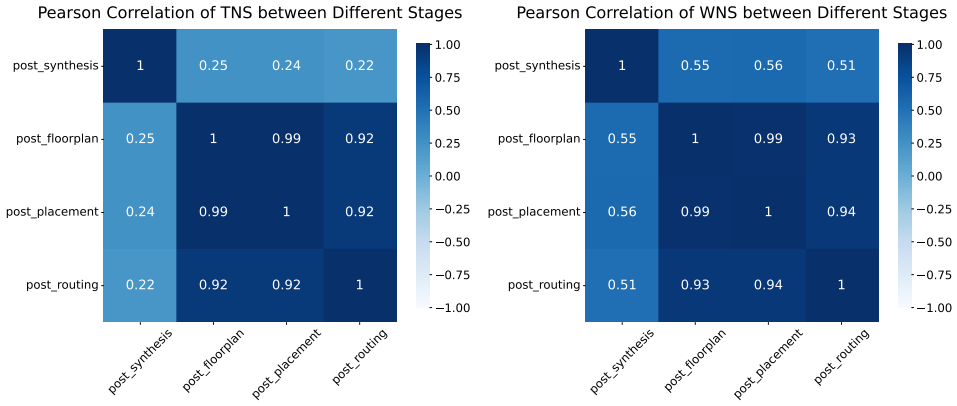


Figure 5: Label Correlation Across Design Stages

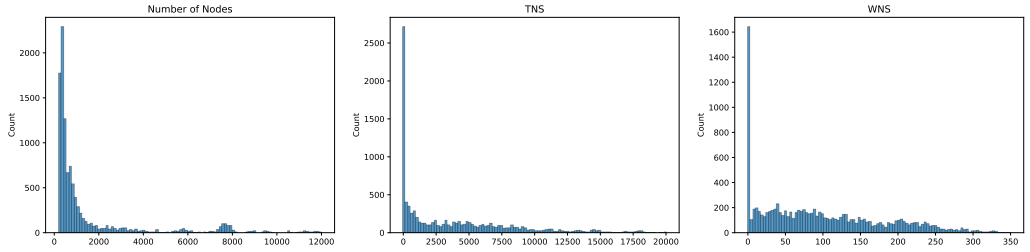


Figure 6: Histogram of Dataset Attributes

Table 3: Node-level input of the timing predictor for the post-synthesis model trained with Direct KD from the PR teacher (held-out test set). Row (iii) is the configuration used throughout the paper and corresponds to Direct KD (PR teacher) in Table 1. MedAE is in the units of the slack labels. Best in bold.

	TNS			WNS		
Node-level representation	MedAE ↓	WMAPE (%) ↓	RSE (%) ↓	MedAE ↓	WMAPE (%) ↓	RSE (%) ↓
(i) GNN output $\mathbf{z}_r$	2201.40 ± 132.98	78.78 ± 2.45	60.46 ± 5.65	55.85 ± 1.66	75.76 ± 1.90	103.59 ± 5.26
(ii) Slack embedding $p(\text{slack}(r))$	495.26 ± 33.28	24.97 ± 2.01	11.48 ± 3.52	15.83 ± 0.94	29.38 ± 2.16	$\mathbf{43.00 \pm 18.45}$
(iii) $\mathbf{z}_r + p(\text{slack}(r))$ (ours)	$\mathbf{399.07 \pm 27.79}$	$\mathbf{24.05 \pm 1.51}$	$\mathbf{11.24 \pm 3.13}$	$\mathbf{15.45 \pm 0.99}$	$\mathbf{28.64 \pm 2.30}$	45.16 ± 18.84

predictor, we add to the node-level representation of each sequential element a learned embedding of its current-stage STA slack,

$$\mathbf{z}_r \leftarrow \mathbf{z}_r + p\big(\text{slack}(r)\big), \tag{10}$$

where $p : \mathbb{R} \to \mathbb{R}^{d_{\text{emb}}}$ is a learnable projector. The term $p(\text{slack}(r))$ acts as a latent bias that passes the slack information directly to the timing predictor. We found that it improves both accuracy and training stability.

Ablation. Table 3 compares three choices of the node-level representation for the post-synthesis model trained with Direct KD from the PR teacher: (i) the GNN output $\mathbf{z}_r$ alone, (ii) the slack embedding $p(\text{slack}(r))$ alone, and (iii) their sum, which we use throughout the paper. In variant (ii), the subgraph- and graph-level representations are unchanged and therefore still use all input features. Removing the slack embedding (i) degrades every metric sharply: TNS WMAPE rises from 24.05% to 78.78%, and WNS RSE exceeds 100%, i.e., the predictions are less accurate than the mean. The current-stage slack is thus the most informative input for predicting post-routing slack. Although it is also one of the GNN's input features, message passing mixes it with the features of neighboring nodes, which is a likely reason the model cannot recover it precisely from $\mathbf{z}_r$; the slack embedding passes it to the timing predictor directly. Conversely, using the slack embedding without the GNN output (ii) comes close to the full model. The sum (iii) is better on five of the six metrics, most clearly on TNS MedAE (399.07 versus 495.26), although the other differences are within one standard deviation. The GNN node representation therefore adds a smaller refinement on top of the slack, mainly for the typical circuit.

D Comparison with Prior Timing Prediction Methods

Table 4 summarizes how STEP-KD differs from the prior timing prediction methods discussed in Section 2.1.

Table 4: Differences between STEP-KD and prior timing prediction methods.

Method	Input stage	Prediction target	Training paradigm
RTL-Timer (Fang et al., 2024b)	RTL (pre-synthesis)	Post-synthesis timing	Supervised
NetTAG (Fang et al., 2025a)	Post-synthesis netlist	Slack and other tasks	Self-supervised pre-training
CircuitFusion (Fang et al., 2025b)	RTL (multimodal)	TNS/WNS and other tasks	Self-supervised pre-training
STEP-KD (ours)	Post-synthesis netlist	Post-routing TNS/WNS	Supervised sequential KD

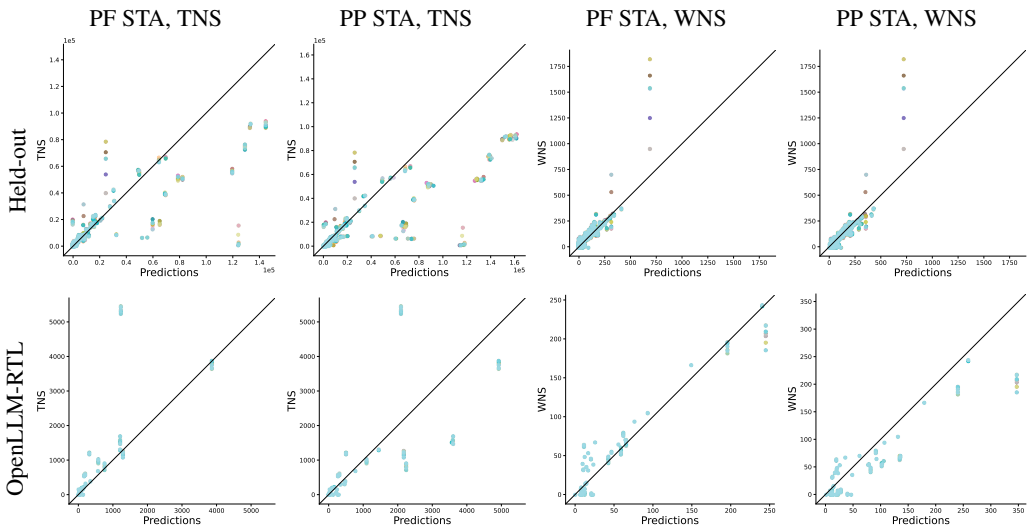


Figure 7: Post-routing TNS and WNS (magnitudes, vertical axis) versus the STA estimate at the post-floorplan (PF) and post-placement (PP) stages (horizontal axis) on the held-out test set (top) and the OpenLLM-RTL benchmark (bottom). The diagonal marks agreement between the intermediate-stage and post-routing STA.

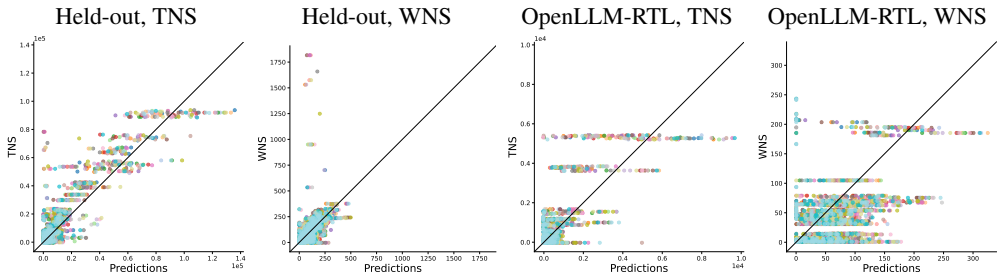


Figure 8: Predicted versus post-routing TNS and WNS (magnitudes) for the No KD baseline, in the same format as Figure 2.

Besides the fact that our core contribution is different from prior work (such as RTL-Timer, NetTAG or CircuitFusion), these prior work also differ from STEP-KD in input stage, prediction target and training paradigm: RTL-Timer and CircuitFusion operate on RTL rather than on the post-synthesis netlist, RTL-Timer predicts post-synthesis rather than post-routing timing, and NetTAG and CircuitFusion rely on self-supervised pre-training and requires pre-training on a large corpus before fine-tuning on downstream tasks.

E Complete Scatter Plots

Figure 7 compares the post-routing TNS and WNS with the STA estimates at the post-floorplan and post-placement stages, and Figure 8 shows the predictions of the No KD baseline as a reference for Figure 2.

Timing changes across design stages. The intermediate-stage STA estimates show that the circuits with the largest post-routing WNS on the held-out set are underestimated at every earlier stage, although the estimates increase along the flow: from about 200 at post-synthesis (Figure 2) to about 700 at the PF and PP stages (Figure 7), against a post-routing WNS of about 950 to 1,820. Their post-routing WNS also differs considerably between implementations that share the same PF and PP estimate.

Table 5: Results of Table 1 reported with R^2 and the mean absolute error (MAE, in the units of the slack labels) on the held-out test set (top) and the OpenLLM-RTL benchmark (bottom). Best per dataset in bold.

	TNS		WNS	
Method	R^2 (%) ↑	MAE ↓	R^2 (%) ↑	MAE ↓
Held-out test set				
STA	-45.49 ± 3.11	5072.45 ± 106.68	53.48 ± 14.72	32.25 ± 2.32
No KD	83.17 ± 4.06	2604.43 ± 180.34	37.86 ± 14.24	39.81 ± 3.06
Direct KD (PR teacher)	88.76 ± 3.13	1619.63 ± 126.85	54.84 ± 18.84	25.74 ± 2.80
Direct KD (PP teacher)	89.05 ± 3.71	1649.69 ± 151.67	54.67 ± 19.40	26.17 ± 2.64
Direct KD (PF teacher)	90.21 ± 3.37	1592.88 ± 130.81	56.97 ± 20.25	26.32 ± 2.80
STEP-KD w/o sequential KD	86.70 ± 3.36	2028.72 ± 148.66	55.04 ± 17.65	24.04 ± 2.28
STEP-KD	$\mathbf{92.15 \pm 2.95}$	$\mathbf{1330.00 \pm 125.66}$	$\mathbf{58.91 \pm 17.81}$	$\mathbf{19.12 \pm 2.24}$
OpenLLM-RTL benchmark				
STA	56.58 ± 0.50	221.75 ± 2.13	83.12 ± 0.68	$\mathbf{17.32 \pm 0.15}$
No KD	47.93 ± 9.63	420.68 ± 21.49	-58.13 ± 7.75	51.03 ± 1.91
Direct KD (PR teacher)	74.92 ± 4.51	232.25 ± 13.96	56.60 ± 2.22	28.07 ± 1.11
Direct KD (PP teacher)	57.96 ± 9.76	264.40 ± 16.85	67.56 ± 2.89	23.25 ± 1.11
Direct KD (PF teacher)	85.14 ± 4.50	202.79 ± 15.89	68.78 ± 1.86	25.85 ± 0.83
STEP-KD w/o sequential KD	88.07 ± 4.43	172.07 ± 14.10	79.92 ± 0.71	18.29 ± 0.31
STEP-KD	$\mathbf{93.51 \pm 1.23}$	$\mathbf{145.60 \pm 7.50}$	$\mathbf{83.48 \pm 0.64}$	18.78 ± 0.35

Sensitivity to implementation parameters. The implementations of a design differ only in their OpenROAD parameters (core utilization and placement density) and are otherwise identical at the post-synthesis stage, yet these parameters largely determine the circuit's timing after the later stages. Post-synthesis STA does not use them, so it predicts the same value for every implementation of a design: its predictions form vertical columns in Figure 2, and the points from the different evaluation runs overlap, even where the post-routing timing differs substantially across implementations. STEP-KD receives these parameters as input features, and its predictions change with them, which appears as horizontal spread and as distinct colors from different runs. Taking the physical design parameters into account is part of STEP-KD's advantage over STA and over prior work that estimates timing before the physical design stages. On OpenLLM-RTL, where the post-routing timing of a design varies little across implementations, this sensitivity also shows up as spread of the predictions around the label, suggesting room to improve how the model uses these parameters. The No KD baseline (Figure 8), which receives the same input features, shows a much wider spread: on OpenLLM-RTL, for example, its TNS predictions for the same design range over several thousand, and several circuits with a large post-routing WNS are predicted to have almost none. Distillation from the later stages therefore makes the use of these parameters considerably more precise.

F Complete Ablation Results

Table 5 reports the results of Table 1 with R^2 and MAE, and Tables 6 and 7 report all metrics for the ablations in Figures 3 and 4.

G Results on OpenLLM-RTL

OpenLLM-RTL main results are reported in Table 1 (bottom) and Figures 3 and 4. Table 8 mirrors Table 2, and Tables 9 and 10 give complete results for the ablations.

Table 6: Complete results for the multi-granularity ablation on the held-out test set (Figure 3). Each variant distills from the same PF, PP and PR teachers as STEP-KD but aligns only the listed granularity; *All* is the full STEP-KD. No KD is included for reference. MAE is in the units of the slack labels. Best in bold.

	TNS				WNS			
KD granularity	R^2 (%) ↑	WMAPE (%) ↓	RSE (%) ↓	MAE ↓	R^2 (%) ↑	WMAPE (%) ↓	RSE (%) ↓	MAE ↓
No KD (reference)	83.17 ± 4.06	38.75 ± 2.20	16.83 ± 4.06	2604.43 ± 180.34	37.86 ± 14.24	44.49 ± 2.23	62.14 ± 14.24	39.81 ± 3.06
Graph only	90.74 ± 2.78	24.07 ± 1.27	9.26 ± 2.78	1617.55 ± 110.02	56.93 ± 16.72	26.53 ± 1.82	43.07 ± 16.72	23.70 ± 2.21
Subgraph only	49.72 ± 3.98	56.36 ± 1.58	50.28 ± 3.98	3786.66 ± 155.30	52.70 ± 14.19	38.69 ± 1.54	47.30 ± 14.19	34.54 ± 2.22
Node only	89.84 ± 3.22	26.95 ± 1.92	10.16 ± 3.22	1812.13 ± 158.15	49.84 ± 16.68	30.62 ± 1.82	50.16 ± 16.68	27.35 ± 2.29
All (STEP-KD)	$\mathbf{92.15 \pm 2.95}$	$\mathbf{19.78 \pm 1.54}$	$\mathbf{7.85 \pm 2.95}$	$\mathbf{1330.00 \pm 125.66}$	$\mathbf{58.91 \pm 17.81}$	$\mathbf{21.39 \pm 1.97}$	$\mathbf{41.09 \pm 17.81}$	$\mathbf{19.12 \pm 2.24}$

Table 7: Complete results for the projection-depth ablation of Direct KD from the PR teacher on the held-out test set (Figure 4). The default Direct KD (PR teacher) in Table 1 uses 3-layer projections. STEP-KD, whose PS model composes three 3-layer projections, is included for reference. Best in bold.

	TNS				WNS			
Projection layers	R^2 (%) ↑	WMAPE (%) ↓	RSE (%) ↓	MAE ↓	R^2 (%) ↑	WMAPE (%) ↓	RSE (%) ↓	MAE ↓
4	88.45 ± 2.89	24.76 ± 1.22	11.55 ± 2.89	1673.53 ± 108.72	51.62 ± 16.35	26.74 ± 2.01	48.38 ± 16.35	24.19 ± 2.45
6	88.46 ± 4.16	25.47 ± 1.78	11.54 ± 4.16	1730.35 ± 159.76	54.50 ± 22.18	27.34 ± 2.76	45.50 ± 22.18	24.82 ± 3.45
8	88.10 ± 3.84	23.81 ± 1.86	11.90 ± 3.84	1625.50 ± 158.01	43.20 ± 18.49	31.93 ± 2.42	56.80 ± 18.49	29.32 ± 3.18
10	90.42 ± 2.91	20.96 ± 1.20	9.58 ± 2.91	1423.49 ± 99.74	48.40 ± 15.69	25.91 ± 1.81	51.60 ± 15.69	23.61 ± 2.18
STEP-KD (3×3 layers)	$\mathbf{92.15 \pm 2.95}$	$\mathbf{19.78 \pm 1.54}$	$\mathbf{7.85 \pm 2.95}$	$\mathbf{1330.00 \pm 125.66}$	$\mathbf{58.91 \pm 17.81}$	$\mathbf{21.39 \pm 1.97}$	$\mathbf{41.09 \pm 17.81}$	$\mathbf{19.12 \pm 2.24}$

Table 8: STA, teacher models and the effect of sequential knowledge distillation (OpenLLM-RTL benchmark). At each intermediate stage we compare STA, a model trained without distillation from later stages, and a model distilled along the design ladder. At PS both STEP-KD variants align to PF, PP and PR teachers one at a time and differ only in whether those teachers were themselves distilled; post-synthesis STA is reported in Table 1. MedAE (median absolute error) is in the units of the slack labels. Best within each stage in bold. †The PR teacher receives post-routing slack as an input feature, so its near-perfect score is expected and shown only for completeness.

		TNS			WNS		
Stage	Predictor	MedAE ↓	WMAPE (%) ↓	RSE (%) ↓	MedAE ↓	WMAPE (%) ↓	RSE (%) ↓
PR	STA	0.00 ± 0.00	0.04 ± 0.00	0.00 ± 0.00	0.00 ± 0.00	0.39 ± 0.02	0.01 ± 0.00
	Teacher†	7.82 ± 0.76	4.11 ± 0.32	0.16 ± 0.04	1.30 ± 0.15	4.61 ± 0.20	0.56 ± 0.04
PP	STA	84.52 ± 7.03	61.25 ± 0.53	47.19 ± 0.67	18.89 ± 0.17	48.28 ± 0.45	34.23 ± 1.27
	Independent (no KD)	$\mathbf{46.65 \pm 2.91}$	$\mathbf{29.55 \pm 1.17}$	$\mathbf{9.86 \pm 0.84}$	$\mathbf{13.00 \pm 0.77}$	$\mathbf{32.59 \pm 0.85}$	$\mathbf{14.13 \pm 0.89}$
	KD from PR	91.96 ± 4.35	40.77 ± 1.32	16.27 ± 1.60	18.68 ± 0.65	44.54 ± 1.08	23.47 ± 1.08
PF	STA	$\mathbf{51.20 \pm 0.43}$	40.18 ± 0.33	43.34 ± 0.48	$\mathbf{9.60 \pm 0.19}$	$\mathbf{27.06 \pm 0.35}$	$\mathbf{10.28 \pm 0.35}$
	Independent (no KD)	87.40 ± 6.52	48.94 ± 1.91	26.31 ± 3.53	22.24 ± 1.40	56.11 ± 2.34	42.83 ± 3.30
	KD from PP (PR→PP)	59.96 ± 4.45	$\mathbf{27.49 \pm 1.81}$	$\mathbf{13.30 \pm 3.25}$	17.57 ± 0.99	35.65 ± 0.84	13.02 ± 0.57
PS	STEP-KD w/o sequential KD	76.74 ± 5.08	32.41 ± 2.67	11.93 ± 4.43	$\mathbf{14.10 \pm 0.71}$	$\mathbf{35.58 \pm 0.61}$	20.08 ± 0.71
	STEP-KD	$\mathbf{62.43 \pm 4.14}$	$\mathbf{27.43 \pm 1.41}$	$\mathbf{6.49 \pm 1.23}$	15.98 ± 0.69	36.55 ± 0.72	$\mathbf{16.52 \pm 0.64}$

Table 9: Complete results for the multi-granularity ablation on the OpenLLM-RTL benchmark (Figure 3). Each variant distills from the same PF, PP and PR teachers as STEP-KD but aligns only the listed granularity; *All* is the full STEP-KD. No KD is included for reference. MAE is in the units of the slack labels. Best in bold.

	TNS				WNS			
KD granularity	R^2 (%) ↑	WMAPE (%) ↓	RSE (%) ↓	MAE ↓	R^2 (%) ↑	WMAPE (%) ↓	RSE (%) ↓	MAE ↓
No KD (reference)	47.93 ± 9.63	79.37 ± 4.03	52.07 ± 9.63	420.68 ± 21.49	-58.13 ± 7.75	99.34 ± 3.75	158.13 ± 7.75	51.03 ± 1.91
Graph only	87.69 ± 1.72	37.60 ± 1.76	12.31 ± 1.72	199.62 ± 9.28	59.42 ± 1.88	51.50 ± 1.32	40.58 ± 1.88	26.47 ± 0.66
Subgraph only	72.15 ± 2.78	65.89 ± 1.94	27.85 ± 2.78	349.78 ± 9.78	42.28 ± 2.67	74.73 ± 1.53	57.72 ± 2.67	38.41 ± 0.73
Node only	87.43 ± 1.09	46.17 ± 1.77	12.57 ± 1.09	245.13 ± 9.72	37.91 ± 2.50	60.66 ± 1.30	62.09 ± 2.50	31.18 ± 0.69
All (STEP-KD)	$\mathbf{93.51 \pm 1.23}$	$\mathbf{27.43 \pm 1.41}$	$\mathbf{6.49 \pm 1.23}$	$\mathbf{145.60 \pm 7.50}$	$\mathbf{83.48 \pm 0.64}$	$\mathbf{36.55 \pm 0.72}$	$\mathbf{16.52 \pm 0.64}$	$\mathbf{18.78 \pm 0.35}$

Table 10: Complete results for the projection-depth ablation of Direct KD from the PR teacher on the OpenLLM-RTL benchmark (Figure 4). The default Direct KD (PR teacher) in Table 1 uses 3-layer projections. STEP-KD, whose PS model composes three 3-layer projections, is included for reference. Best in bold.

	TNS				WNS			
Projection layers	R^2 (%) ↑	WMAPE (%) ↓	RSE (%) ↓	MAE ↓	R^2 (%) ↑	WMAPE (%) ↓	RSE (%) ↓	MAE ↓
4	88.02 ± 2.49	34.08 ± 2.42	11.98 ± 2.49	180.95 ± 12.68	54.56 ± 2.49	53.96 ± 1.61	45.44 ± 2.49	27.72 ± 0.83
6	52.43 ± 5.18	51.66 ± 1.56	47.57 ± 5.18	274.15 ± 8.13	57.36 ± 1.42	53.70 ± 1.16	42.64 ± 1.42	27.61 ± 0.60
8	38.41 ± 8.33	60.24 ± 2.58	61.59 ± 8.33	320.05 ± 13.70	39.30 ± 3.16	67.33 ± 1.63	60.70 ± 3.16	34.62 ± 0.87
10	91.42 ± 0.52	32.64 ± 0.64	8.58 ± 0.52	173.11 ± 3.57	68.97 ± 1.03	41.95 ± 0.72	31.03 ± 1.03	21.56 ± 0.33
STEP-KD (3×3 layers)	$\mathbf{93.51 \pm 1.23}$	$\mathbf{27.43 \pm 1.41}$	$\mathbf{6.49 \pm 1.23}$	$\mathbf{145.60 \pm 7.50}$	$\mathbf{83.48 \pm 0.64}$	$\mathbf{36.55 \pm 0.72}$	$\mathbf{16.52 \pm 0.64}$	$\mathbf{18.78 \pm 0.35}$