HLS-Timer: Fine-Grained Path-Level Timing Estimation for High-Level Synthesis

Zibo Hu[1], Zhe Lin[2], Renjing Hou[1], Xingyu Qin[1], Jianwang Zhai[1], and Kang Zhao[1]

{huzibo, hourenjing, qinxingyu, zhaijw, zhaokang}@bupt.edu.cn,

linzh235@mail.sysu.edu.cn

[1]Beijing University of Posts and Telecommunications

[2]Sun Yat-sen University

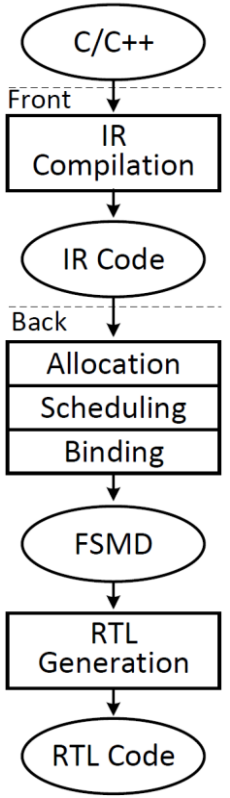
Contents

- Introduction
- Preliminaries
- Methodology
- Evaluation
- Conclusion

Introduction

High-Level Synthesis

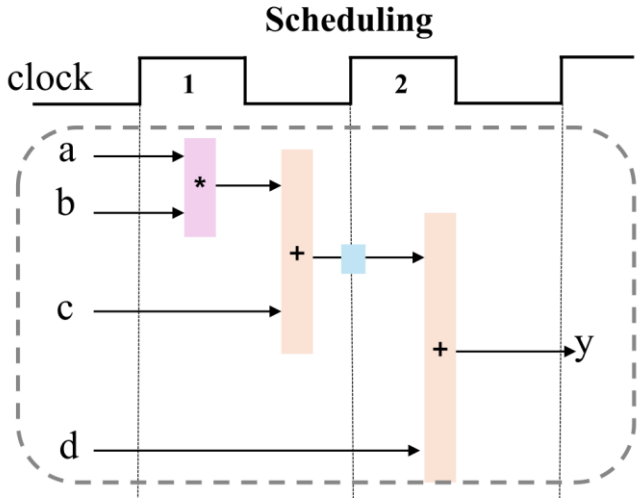
- C/C++ → hardware description languages
- Front-end
  - Intermediate representation (IR) generation
  - Bitwidth reduction, loop unrolling, etc
- Back-end
  - Resource allocation, scheduling, binding
  - Register-transfer level (RTL) code generation
- Directives to tune latency/resource



Motivation: Scheduling in HLS

- **Scheduling**
  ---> Assigning operations to clock cycles based on timing.
- **Current Practice**
  ---> Relying on **fixed latency estimates**
- **Consequence**
  ---> suboptimal scheduling or failed design targets.

```
int func (char a, char b, char c, char d)
{
    char x;
    x=a * b + c + d;
    return x;
}
```



Related Work

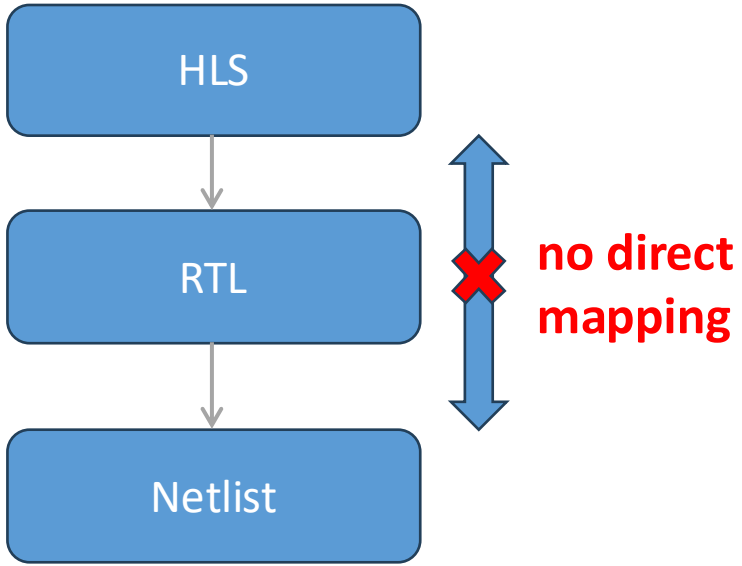
- **Initial Efforts** [1, 2]:
  Focused on high-level metrics (Area), but lacked timing precision.
- **ML/GNN-based HLS Prediction** [3, 4, 5]:
  Advancement: Predicted Critical Path (CP) delays directly from HLS.
  Limitation: **Coarse-grained**; no explicit mapping to physical STA metrics (like path-level slack).
- **Netlist & Layout-Level Prediction** [10, 11, 12, 13]:
  Advancement: High accuracy using GNNs/Transformers.
  Limitation: **Too late**; requires post-synthesis, limiting early-stage design exploration.

TABLE I Comparison of ML-based approaches for High-Level Synthesis (HLS) Timing prediction tasks.

Work	ML model		Target		Task	Feature Source	Tool
	Graph	Non-Graph	FPGA	ASIC			
Pyramid [3]		✓	✓		Resource Usage and Timing	HLS reports	Vivado HLS
Wu et al. [4]	✓		✓		Resource Usage and CP Timing	IR operator information and HLS reports	Vitis HLS
Dai et al. [2]		✓	✓		Resource Usage and Timing	HLS reports	Vivado HLS
De et al. [14]	✓	✓		✓	Timing	HLS reports	Stratus HLS
This Work	✓	✓	✓		**Data delay**, WNS and CP Timing	IR operator information and bind result	Vitis HLS

Challenges

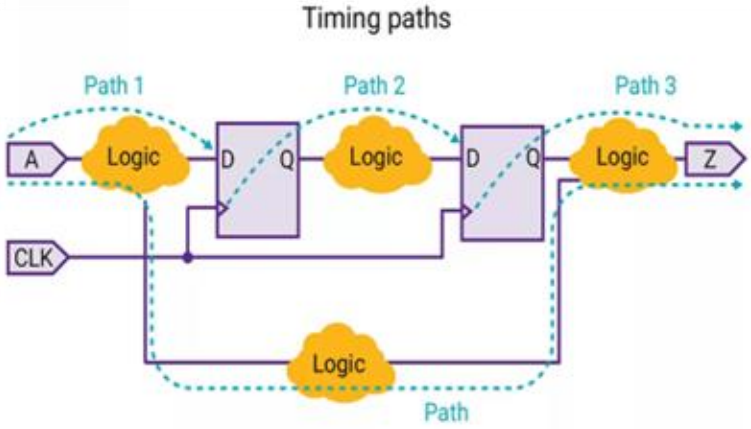
1. ***Structural Gap (Mapping Ambiguity)***
   - **HLS-to-Gate Disconnect**
   - **Information Loss**
     - no feedback
2. ***Multi-Dimensional Complexity***
   - **Physical Factors**
     - layout-dependent effects (e.g., interconnect delays)
     - process variations
   - **Design structure**
     - architectural decisions
     - physical implementation



Preliminaries

Basic Concepts

1. **Timing Path：**
   Signal propagation route from a Source to a Destination endpoint.
2. **Endpoint：**
   Typically Registers or I/O Ports.
3. **Data Delay：**
   Gate Delay + Wire Delay
4. **Static Timing Analysis(STA)：** A stimulus-independent method to verify timing



In the example, each logic cloud represents a combinational logic network. Each path starts at a data launch point, passes through some combinational logic, and ends at a data capture point.

Path	Startpoint	Endpoint
Path 1	Input port	Data input of a sequential element
Path 2	Clock pin of a sequential element	Data input of a sequential element
Path 3	Clock pin of a sequential element	Output port
Path 4	Input port	Output port

Slack = Required Arrival Time - Arrival Time

Problem Formulation

Problem 1. (Data delay of timing path estimation)

$$\forall p_i \in H, M_{Dt}(p_i) \rightarrow Dt_{\mathrm{N}}(p_i)$$

Problem 2. (CP and WNS estimation)

$$M_{CP}(Dt) \rightarrow \mathrm{CP}$$
$$M_{WNS}(Dt) \rightarrow \mathrm{WNS}$$

- $\boldsymbol{H}$: HLS Design (Input Space)
- $\boldsymbol{V}$:Synthesized HDL Code
- **N**: Gate-level Netlist (Ground Truth Source)
- $\boldsymbol{p_i} \in \boldsymbol{H}$: A specific timing path sampled from the HLS stage.

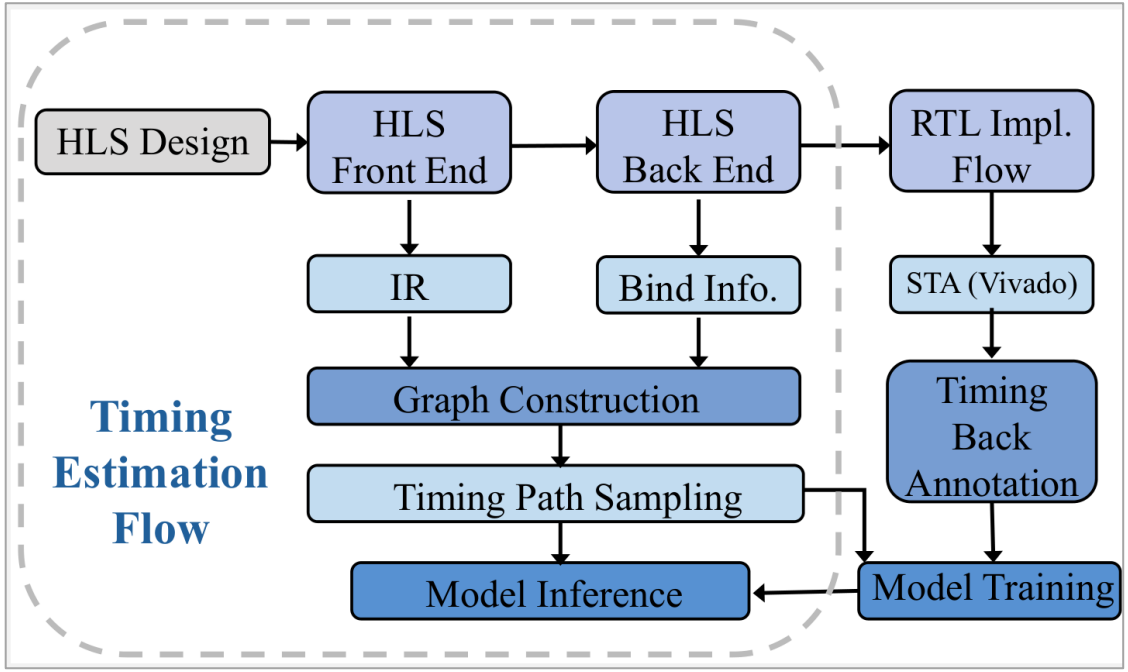
Note: Ground truth is obtained from STA reports on Netlist via our proposed two-stage back-anotation

Methodology

HLS-Timer Framework

Contribution

- Graph Construction with 4 Augmentation Strategies
- Dataflow-driven Recursive Path Sampling
- Two-phase Timing Back-annotation
- Hierarchical GNN-Transformer Estimation Model



Graph Construction

Efficient Graph Construction Flow via Low-Coupling Enhancement Strategies

1.**Register Insertion** 2.**Interface Refactoring** 3.**Submodule Expansion** 4.**Direction Rebuilding**

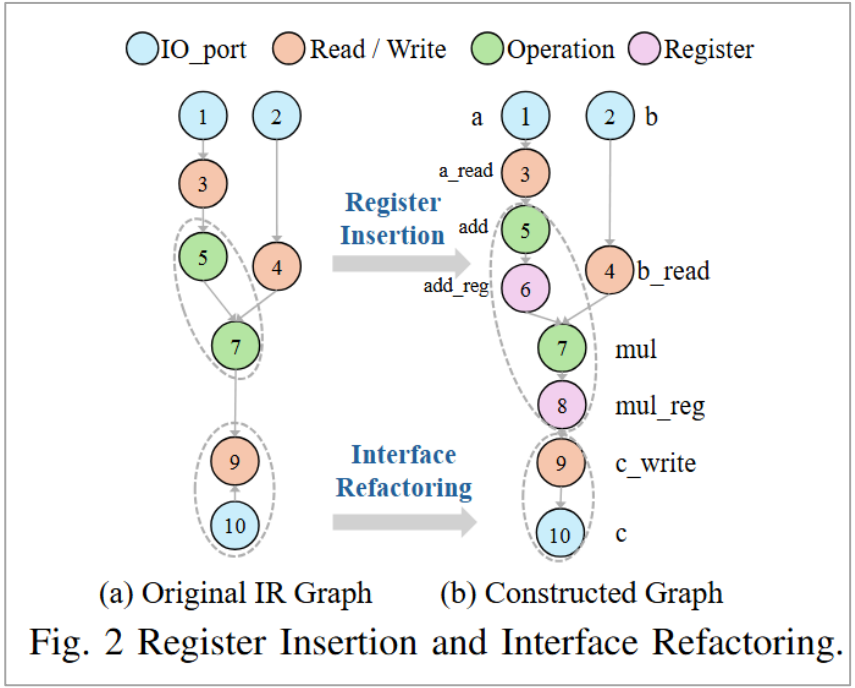


Fig. 2 Register Insertion and Interface Refactoring.

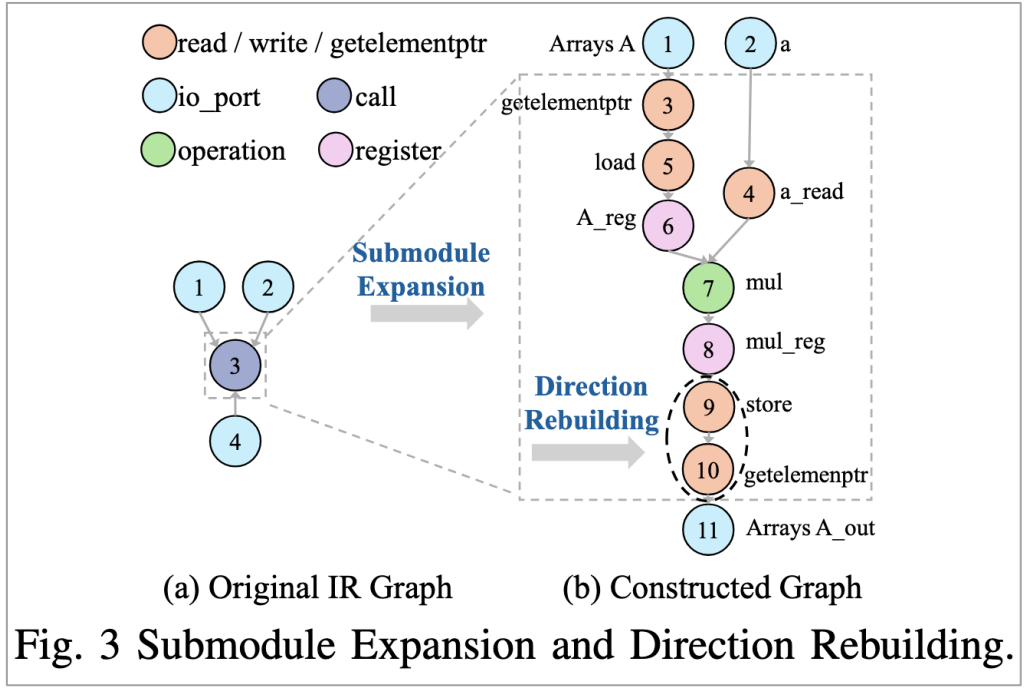


Fig. 3 Submodule Expansion and Direction Rebuilding.

Timing Path Sampling

Dataflow-driven Recursive Path Sampling

- **Objective**
  - Systematic extraction of timing paths from HLS-specific graphs.
- **Strategy**
  - Iterative Frontier Expansion (based on DFS).

Algorithm 1 Timing Path Sampling (Simplified Main Procedure)

Input: Graph G, Initial Start Nodes $S_{initial}$
Output: All Timing Paths P_{all}

procedure `Timing Path Sampling`(G, $S_{initial}$)
  $P_{all} \leftarrow \varnothing$
  $S_{current} \leftarrow S_{initial}$ // List of start nodes for DFS in the current iteration
  $S_{processed\_starts} \leftarrow \varnothing$ // Set of all nodes from which DFS has been initiated
  while $S_{current}$ is not empty **do**
    $P_{round} \leftarrow \varnothing$ // Timing paths newly discovered in this round
    $S_{next\_potential\_starts} \leftarrow \varnothing$ // Set of potential start nodes for the next iteration
    for each node s in $S_{current}$ **do**
      if s is not in $S_{processed\_starts}$ **then**
        `dfs_path_sampling`(G, s, P_{round}) // Call DFS function to explore paths starting from s and add them to P_{round}
        Add s to $S_{processed\_starts}$
        for each path p in P_{round} that started from s **do**
          Add `get_end_node`(p) to $S_{next\_potential\_starts}$
        end for
      end if
    end for
    $P_{all} \leftarrow P_{all} \cup P_{round}$
    $S_{current} \leftarrow S_{next\_potential\_starts} \setminus S_{processed\_starts}$ // Update $S_{current}$ with new, unprocessed start nodes
  end while
  return P_{all}
end procedure

Example

Process：

1. **Initialization**: Start from I/O ports.
2. **Traversal**: Execute DFS to capture paths from the current frontier.
3. **Update**: Set new endpoints as start nodes for the next round.
4. **Termination**: Stop when no new start nodes are discovered.

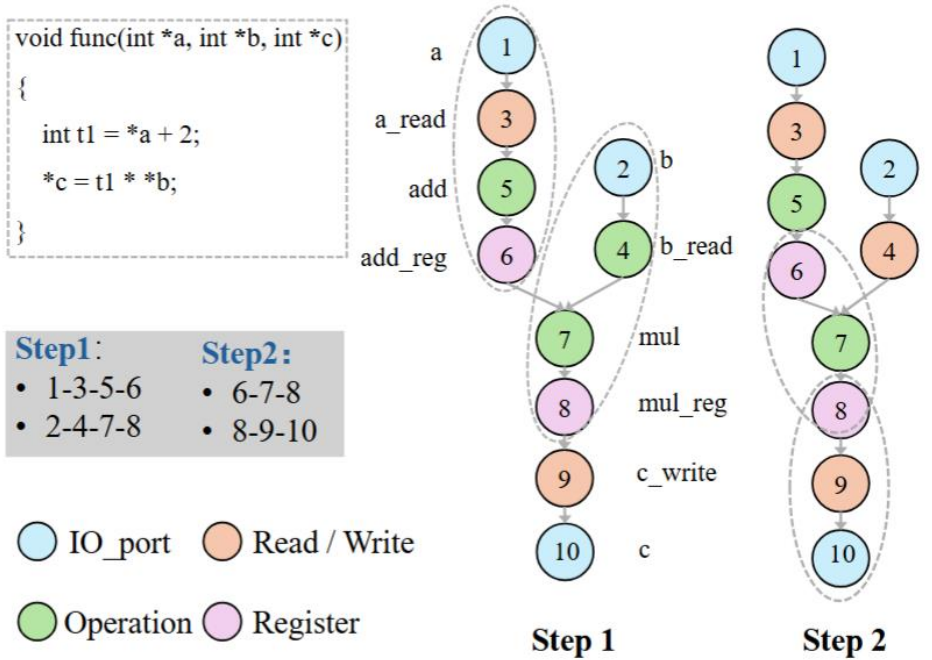


Fig. 4 Example of Timing Path Sampling.

Two-phase Back-annotation

Phase 1:

Netlist to RTL (Bit-level to Word-level)

- **The Challenge**: Netlist are bit-level, while RTL uses word-level.
- **Mapping Strategy**: Inspired by STA timing propagation.
- **The "Maximum" Principle**:

$$D_{\text{RTL}}(S,T) = \max_{r_i \in R} D_{\text{Netlist}}(s_i, t_i)$$

Objective: Achieve precise timing label alignment from Gate-level STA to HLS IR.

Two-phase Back-annotation

Phase 2:

RTL to HLS IR (Name Mapping)

- **Bridging the Gap**: Leverages HLS front-end and back-end binding information.
- **Consistency**: IR nodes maintain names largely consistent with original C code.
- **Bidirectional Mapping**: Establishing links between IR-level operations and their synthesized RTL component names.

Hierarchical Estimation Model

- **Node Feature**
  - as shown in TABLE III
- **Tasks/Labels**
  - Path-level：Data delay
  - Design-level：CP、WNS
- **Dataset**
  - Synthetic: **12,653** HLS-compliant C programs (automated generation).
  - Real-world: **39** designs from *MachSuite* and *PolyBench/C*.

TABLE III Node Features and Example Values.

Feature	Description	Values
Node category	General node type	port, operation node, reg, etc.
Bitwidth	Bitwidth of the node	0∼256, misc
Opcode category	Opcode categories	binary_unary, bitwise, memory, etc.
Opcode	Opcode of the node	load, add, mul, store, etc.
Is start of path	Whether the node is the starting node of a path	0, 1, misc
Is LCDNode	Whether the node is LCD Node	0, 1, misc
Cluster group	Cluster number of the node	-1∼256, misc

Hierarchical Estimation Model

- **Model**
  - GNNs: GCN、GAT、GIN、GraphSAGE
  - Transformer
  - MLP

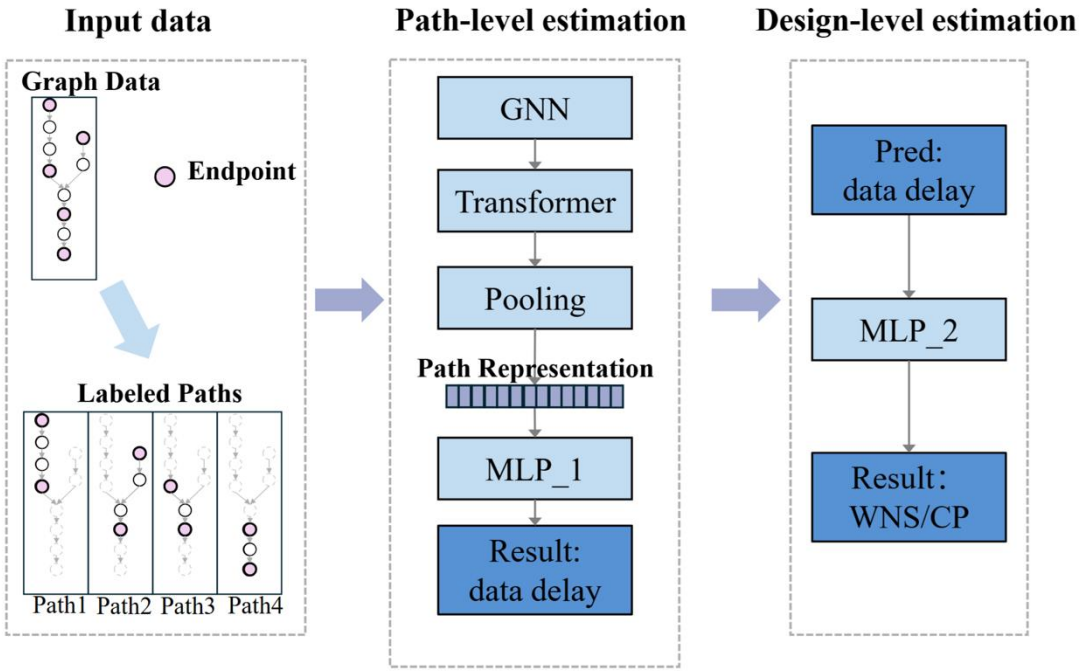


Fig. 5 Hierarchical Estimation Model

Evaluation

Experiments Settings

- **Framework & Tools**: Python/PyTorch implementation using Vitis & Vivado 2022.1 for ground-truth labeling.
- **Model Architecture**: Hybrid 2-layer GNN + 2-layer Transformer (4 heads) with a 64-64-1 MLP.
- **Dataset Strategy**: 80/10/10 split (Train/Val/Test); real-case benchmarks used specifically for generalization testing.
- **Training Profile**: Optimized via Adam for 100 epochs.
- **Baseline:** DAC22[1]
- **Evaluation Metrics**: R, R^2, MAPE

---

[1]Wu, H. Yang, Y. Xie, P. Li, and C. Hao, “High-level synthesisperformance prediction using gnns: Benchmarking, modeling, andadvancing,” DAC, 2022

Evaluation

Path-level Performance

- As shown in Table IV, our method achieves state-of-the-art performance in fine-grained data delay regression.

Design-level Performance

- **Higher Accuracy**: Outperformed baseline in **CP** (Critical Path) estimation (**6.79%** MAPE vs. 7.68%).
- **50% Training Time Reduction**: Achieved through our lightweight architecture compared to the baseline's deep model.
- **Multi-task Support**: Simultaneous prediction of **WNS** metrics (9.97% MAPE), which the baseline does not support.

TABLE IV Path-level timing accuracy and ablation study.

Fine-Grained	Method	R	R^2	MAPE (%)
data delay	MLP	0.9599	0.9196	20.8682
	Transfomer	0.9560	0.9115	22.3479
	GCN	0.9560	0.9131	25.4475
	GAT	0.9522	0.9055	26.1010
	GIN	0.9574	0.9148	22.5104
	GraphSage	0.9566	0.9126	22.2962
	This Work	0.9416	**0.9273**	**18.9554**

TABLE V MAPE of graph-level indicators CP/WNS

Metric	Method	MAPE(%)
CP	Wu et al. [4]	7.6805
	This Work	**6.7926**
WNS	This Work	**9.9730**

Evaluation

- Extensive evaluation on 20 diverse real-world cases (spanning linear algebra, stencil computation, and data streaming) to **validate practical robustness**.
  - Consistent accuracy across diverse benchmarks with minimal deviation from post-synthesis ground truth.

- **Zero-overhead timing estimation**: Direct path-level prediction for any Vitis-HLS compliant design.

TABLE VI Performance on Real-word designs.

design	Path-level Pred		Design-level Pred					
	data delay		CP			WNS		
	MAPE(%)	R	Real(ns)	Pred(ns)	MAPE(%)	Real(ns)	Pred(ns)	MAPE(%)
aA	9.020	1.000	10.235	10.208	0.261	-4.092	-4.302	5.139
aAB	18.036	0.962	11.474	10.210	11.017	-4.091	-4.431	8.307
aAplusbB	16.748	0.988	10.827	10.834	0.062	-4.136	-4.369	5.628
AB_1	13.659	0.978	12.021	10.058	16.302	-4.445	-4.510	1.460
AB_2	11.507	0.969	10.574	10.074	4.725	-4.176	-4.375	4.777
ABplusC	11.944	0.983	11.282	10.044	10.974	-4.274	-4.298	0.560
ABplusCD	15.866	0.956	11.630	9.925	14.657	-3.961	-4.187	5.697
ABx_1	27.288	1.000	11.168	9.923	11.144	-7.195	-5.652	21.441
ABx_2	17.733	0.999	11.336	9.719	14.267	-6.012	-4.455	25.906
AplusB	9.523	1.000	11.237	10.465	6.866	-4.192	-4.217	0.587
bfs	23.833	0.821	11.239	11.131	0.957	-6.683	-4.349	34.901
bicg	7.917	1.000	8.692	8.267	4.888	-5.244	-4.405	15.998
gemm	18.815	0.978	11.470	10.285	10.334	-4.420	-4.438	0.402
k3mm	16.937	0.963	11.783	9.522	19.192	-4.217	-4.375	3.744
mac_1	12.467	0.975	11.116	10.868	2.226	-4.445	-4.418	0.610
mac_2	8.941	0.995	10.820	10.972	1.406	-4.444	-4.443	0.014
mac_3	22.463	0.977	11.257	10.697	4.972	-4.406	-4.383	0.528
mac_4	25.101	0.976	11.239	10.684	4.937	-4.465	-4.488	0.507
mac_5	9.229	0.999	10.300	10.409	1.055	-4.492	-4.490	0.040
syrk	20.162	0.927	10.784	10.341	4.112	-4.263	-4.430	3.928
Avg	**15.859**	**0.972**	**11.024**	**10.232**	**7.218**	**-4.683**	**-4.451**	**7.009**

Conclusion

Conclusion

- This paper introduces HLS-Timer, the first fine-grained path-level timing prediction framework for HLS designs.
  - HLS-Timer accurately predicts timing path data delays by leveraging local and global information through novel graph construction and back-annotation.
  - Experimental results show its high accuracy, efficiency, and portability, positioning it as a promising tool for HLS timing optimization.
- **Future work** will focus on
  - enhancing prediction accuracy and
  - using these capabilities to optimize HLS scheduling algorithms.

THANK YOU!