

REPRODUCIBILITY STATEMENT

We provide references to relevant sections and materials to assist readers and researchers in replicating our results.

Dataset description: All datasets used in our experiments are from open-source benchmarks. A summary of these datasets is available in Appendix A, with a demonstration example shown in Appendix B.2. Detailed preprocessing methods are described in Appendix B, including the different circuit modality generation, sub-circuit generation, and downstream task label collection. The corresponding scripts can be found in our open-source repository.

Open access to CircuitFusion code: The source code for CircuitFusion is publicly available at: <https://anonymous.4open.science/r/CircuitFusion-EB45>. The repository includes scripts with step-by-step instructions to replicate the primary results presented in this paper.

REFERENCES

- NanGate 45nm Open Cell Library*. <https://si2.org/open-cell-library/>, a.
- OpenCores: The reference community for Free and Open Source gateway IP cores*. <https://opencores.org/>, b.
- Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altschmidt, Sam Altman, Shyamal Anadkat, et al. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.
- Hassan Akbari, Liangzhe Yuan, Rui Qian, Wei-Hong Chuang, Shih-Fu Chang, Yin Cui, and Boqing Gong. Vatt: Transformers for multimodal self-supervised learning from raw video, audio and text. *Advances in Neural Information Processing Systems*, 34:24206–24221, 2021.
- Alon Amid, David Biancolin, Abraham Gonzalez, Daniel Grubb, Sagar Karandikar, Harrison Liew, Albert Magyar, Howard Mao, Albert Ou, Nathan Pemberton, et al. Chipyard: Integrated design, simulation, and implementation framework for custom socs. *IEEE Micro*, 40(4), 2020.
- Chen Bai, Jiayi Huang, Xuechao Wei, Yuzhe Ma, Sicheng Li, Hongzhong Zheng, Bei Yu, and Yuan Xie. Archexplorer: Microarchitecture exploration via bottleneck analysis. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 268–282, 2023.
- Hangbo Bao, Wenhui Wang, Li Dong, Qiang Liu, Owais Khan Mohammed, Kriti Aggarwal, Subhojit Som, Songhao Piao, and Furu Wei. Vlmo: Unified vision-language pre-training with mixture-of-modality-experts. *Advances in Neural Information Processing Systems*, 35:32897–32912, 2022.
- Robert Brayton and Alan Mishchenko. Abc: An academic industrial-strength verification tool. In *Computer Aided Verification: 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings 22*, pp. 24–40. Springer, 2010.
- Fulvio Corno, Matteo Sonza Reorda, and Giovanni Squillero. Rt-level itc’99 benchmarks and first atpg results. *IEEE Design & Test of computers (ITC)*, 2000.
- Chenhui Deng, Zichao Yue, Cunxi Yu, Gokce Sarar, Ryan Carey, Rajeev Jain, and Zhiru Zhang. Less is more: Hop-wise graph attention for scalable and generalizable learning on circuits. *arXiv preprint arXiv:2403.01317*, 2024.
- Jacob Devlin. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.
- Alexey Dosovitskiy. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.
- Yufan Du, Zizheng Guo, Xun Jiang, Zhuomin Chai, Yuxiang Zhao, Yibo Lin, Runsheng Wang, and Ru Huang. Powpredict: Cross-stage power prediction with circuit-transformation-aware learning. In *Proceedings of 2024 ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6. ACM, 2024.

- 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 2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–9. IEEE, 2023.
- Wenji Fang, Shang Liu, Hongce Zhang, and Zhiyao Xie. Annotating slack directly on your verilog: Fine-grained rtl timing evaluation for early optimization. In *Proceedings of 2024 ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6. ACM, 2024a.
- Wenji Fang, Yao Lu, Shang Liu, Qijun Zhang, Ceyu Xu, Lisa Wu Wills, Hongce Zhang, and Zhiyao Xie. Transferable pre-synthesis ppa estimation for rtl designs with data augmentation techniques. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2024b.
- Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, et al. Codebert: A pre-trained model for programming and natural languages. *arXiv preprint arXiv:2002.08155*, 2020.
- Difei Gao, Ke Li, Ruiping Wang, Shiguang Shan, and Xilin Chen. Multi-modal graph neural network for joint reasoning on vision and scene text. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 12746–12756, 2020.
- Daya Guo, Shuai Lu, Nan Duan, Yanlin Wang, Ming Zhou, and Jian Yin. Unixcoder: Unified cross-modal pre-training for code representation. *arXiv preprint arXiv:2203.03850*, 2022.
- Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.
- Zhenyu Hou, Xiao Liu, Yukuo Cen, Yuxiao Dong, Hongxia Yang, Chunjie Wang, and Jie Tang. Graphmae: Self-supervised masked graph autoencoders. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 594–604, 2022.
- Chankyu Lee, Rajarshi Roy, Mengyao Xu, Jonathan Raiman, Mohammad Shoeybi, Bryan Catanzaro, and Wei Ping. Nv-embed: Improved techniques for training llms as generalist embedding models. *arXiv preprint arXiv:2405.17428*, 2024.
- Junnan Li, Ramprasaath Selvaraju, Akhilesh Gotmare, Shafiq Joty, Caiming Xiong, and Steven Chu Hong Hoi. Align before fuse: Vision and language representation learning with momentum distillation. *Advances in neural information processing systems*, 34:9694–9705, 2021.
- Junnan Li, Dongxu Li, Caiming Xiong, and Steven Hoi. Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation. In *International conference on machine learning*, pp. 12888–12900. PMLR, 2022a.
- Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *International conference on machine learning*, pp. 19730–19742. PMLR, 2023a.
- Min Li, Sadaf Khan, Zhengyuan Shi, Naixing Wang, Huang Yu, and Qiang Xu. Deepgate: Learning neural representations of logic gates. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pp. 667–672, 2022b.
- Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, et al. Starcoder: may the source be with you! *arXiv preprint arXiv:2305.06161*, 2023b.
- Mingjie Liu, Nathaniel Pinckney, Brucek Khailany, and Haoxing Ren. VerilogEval: Evaluating large language models for verilog code generation. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–8. IEEE, 2023a.
- Shang Liu, Wenji Fang, Yao Lu, Qijun Zhang, Hongce Zhang, and Zhiyao Xie. Rtlcoder: Outperforming gpt-3.5 in design rtl generation with our open-source dataset and lightweight solution. *arXiv preprint arXiv:2312.08617*, 2023b.

- Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nazi, et al. A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212, 2021.
- Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. Mteb: Massive text embedding benchmark. *arXiv preprint arXiv:2210.07316*, 2022.
- Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.
- Zehua Pei, Huiling Zhen, Mingxuan Yuan, Yu Huang, and Bei Yu. Betterv: Controlled verilog generation with discriminative guidance. In *Forty-first International Conference on Machine Learning (ICML)*, 2024.
- Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pp. 8748–8763. PMLR, 2021.
- Martin Rapp, Hussam Amrouch, Yibo Lin, Bei Yu, David Z Pan, Marilyn Wolf, and Jörg Henkel. Mlcad: A survey of research in machine learning for cad keynote paper. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(10):3162–3181, 2021.
- Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: Visual explanations from deep networks via gradient-based localization. In *Proceedings of the IEEE international conference on computer vision*, pp. 618–626, 2017.
- Prianka Sengupta, Aakash Tyagi, Yiran Chen, and Jiang Hu. How good is your verilog rtl code? a quick answer from machine learning. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, pp. 1–9, 2022.
- Zhengyuan Shi, Hongyang Pan, Sadaf Khan, Min Li, Yi Liu, Junhua Huang, Hui-Ling Zhen, Mingxuan Yuan, Zhufei Chu, and Qiang Xu. Deepgate2: Functionality-aware circuit representation learning. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–9. IEEE, 2023.
- Atefeh Sohrabizadeh, Yunsheng Bai, Yizhou Sun, and Jason Cong. Robust gnn-based representation learning for hls. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–9. IEEE, 2023.
- Shobha Vasudevan, Wenjie Joe Jiang, David Bieber, Rishabh Singh, C Richard Ho, Charles Sutton, et al. Learning semantic representations to verify hardware designs. *Advances in Neural Information Processing Systems*, 34:23491–23504, 2021.
- VexRiscv. VexRiscv: A FPGA friendly 32 bit RISC-V CPU implementation, 2022. URL <https://github.com/SpinalHDL/VexRiscv>.
- Minjie Yu Wang. Deep graph library: Towards efficient and scalable deep learning on graphs. In *ICLR workshop on representation learning on graphs and manifolds*, 2019.
- Yue Wang, Hung Le, Akhilesh Deepak Gotmare, Nghi DQ Bui, Junnan Li, and Steven CH Hoi. Codet5+: Open code large language models for code understanding and generation. *arXiv preprint arXiv:2305.07922*, 2023.
- Ziyi Wang, Chen Bai, Zhuolun He, Guangliang Zhang, Qiang Xu, Tsung-Yi Ho, Bei Yu, and Yu Huang. Functionality matters in netlist representation learning. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, pp. 61–66, 2022.
- Clifford Wolf, Johann Glaser, and Johannes Kepler. Yosys-a free verilog synthesis suite. In *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*, volume 97, 2013.

- Ceyu Xu, Chris Kjellqvist, and Lisa Wu Wills. SNS's not a synthesizer: a deep-learning-based synthesis predictor. In *Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA)*, pp. 847–859, 2022.
- Ceyu Xu, Pragya Sharma, Tianshu Wang, and Lisa Wu Wills. Fast, robust and transferable prediction for hardware logic synthesis. In *Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pp. 167–179, 2023.
- Shuwen Yang, Zhihao Yang, Dong Li, Yingxueff Zhang, Zhanguang Zhang, Guojie Song, and Jianye Hao. Versatile multi-stage graph neural network for circuit representation. *Advances in Neural Information Processing Systems*, 35:20313–20324, 2022.
- Yongjing Yin, Fandong Meng, Jinsong Su, Chulun Zhou, Zhengyuan Yang, Jie Zhou, and Jiebo Luo. A novel graph-based multi-modal fusion encoder for neural machine translation. *arXiv preprint arXiv:2007.08742*, 2020.
- Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. Do transformers really perform badly for graph representation? *Advances in neural information processing systems*, 34:28877–28888, 2021.
- Cunxi Yu, Houping Xiao, and Giovanni De Micheli. Developing synthesis flows without human knowledge. In *Proceedings of the 55th Annual Design Automation Conference*, pp. 1–6, 2018.
- Dejiao Zhang, Wasi Ahmad, Ming Tan, Hantian Ding, Ramesh Nallapati, Dan Roth, Xiaofei Ma, and Bing Xiang. Code representation learning at scale. *arXiv preprint arXiv:2402.01935*, 2024.
- Hongyi Zhang. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412*, 2017.
- Su Zheng, Lancheng Zou, Peng Xu, Siting Liu, Bei Yu, and Martin Wong. Lay-net: Grafting netlist knowledge on layout-based congestion prediction. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–9. IEEE, 2023.
- Keren Zhu, Hao Chen, Walker J Turner, George F Kokai, Po-Hsuan Wei, David Z Pan, and Haoxing Ren. Tag: Learning circuit spatial embedding from layouts. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–9. IEEE, 2022.

A MORE ON CIRCUIT HDL DATASET

This section provides an overview of the various circuit HDL datasets used in our work, including ITC’99, OpenCores, VexRiscv, and Chipyard. These datasets offer diverse designs that span a range of hardware implementations, enabling comprehensive benchmarking of CircuitFusion across different circuit design tasks.

A.0.1 ITC’99

The ITC’99 benchmark suite (Corno et al., 2000) is a widely used collection of hardware circuit designs, primarily designed for logic synthesis and verification. ITC’99 provides diverse designs ranging from simple combinational logic to more complex sequential circuits. VHDL

A.0.2 OPENCORES

The OpenCores repository (URL, b) offers open-source hardware designs, including a wide variety of digital systems, such as CPUs, memory controllers, communication protocols, etc. OpenCores is a rich dataset for benchmarking HDL models because of its diverse collection of designs, which range from small, simple circuits to large, complex ones. Its open-source nature allows for flexibility in circuit modification, making it ideal for research and development in hardware design.

A.0.3 VEXRISCV

VexRiscv (VexRiscv, 2022) is an open-source, RISC-V compliant CPU core designed using Spinal-HDL. This dataset focuses on CPU design and features a highly configurable architecture, allowing for variations in pipeline stages, instruction sets, and optimizations. The VexRiscv dataset is particularly useful for testing the scalability and flexibility of models in handling CPU-level design tasks, making it a valuable resource for benchmarking models like CircuitFusion on processor design tasks.

A.0.4 CHIPYARD

Chipyard (Amid et al., 2020) is a comprehensive framework for building RISC-V-based system-on-chip (SoC) designs. It includes a collection of CPU cores, accelerators, memory systems, and I/O components, offering a complete design ecosystem for hardware developers. The Chipyard dataset enables testing at the SoC level, providing a broad set of circuits with varying complexities and design objectives.

B MORE ON CIRCUIT DATA PREPROCESSING

B.1 DATASET COLLECTION

In the open-source benchmarks, the HDL code of RTL circuits is provided, where the RTL stage describes the functional behaviors of the circuit. We then use the EDA tool Synopsys Design Compiler® to automatically synthesize the RTL circuits into gate-level netlists. The netlists represent real circuit implementations, consisting of logic gates (e.g., ADD, INV, AND, etc.) and registers (DFF). We employ the open-source NanGate 45nm technology library (URL, a) for the synthesis process. The design quality metrics of netlists are obtained through Synopsys Prime Time® after synthesis, including slack of each register, WNS, TNS, total power, and total area.

B.2 MULTIMODAL AND MULTI-STAGE CIRCUIT: A CASE STUDY

In this subsection, we provide a detailed example demonstrating the three modalities of RTL circuits.

B.2.1 HDL CODE

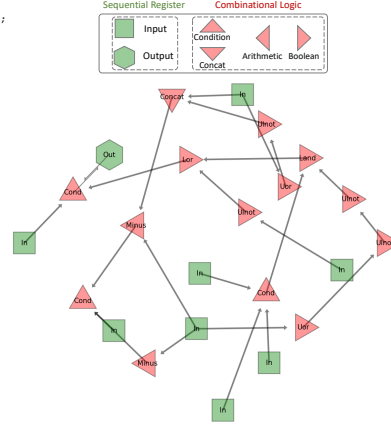
As shown in Figure 7a, the HDL code for each sub-circuit is directly used as one of the input modalities, capturing the functional description of the circuit’s behavior at the RTL stage. In this Verilog HDL code, a `module` represents an entire sub-circuit, where `input` and `output` specify

```

810 module coi (clk, rst, shift_cnt, shift_tip, shift_data, wb_addr_i,
811            cigen_pos_edge, ctrl, cigen_neg_edge, wb_stb_i, wb_cyc_i, shift_s_out);
812     input clk, rst, cigen_neg_edge, cigen_pos_edge, ctrl;
813     input [4:0] wb_addr_i;
814     input shift_cnt, shift_data, shift_tip, wb_cyc_i, wb_stb_i;
815     output reg shift_s_out;
816     wire _008_, _019_, _056_, _088_, _093_, _096_, _097_, _177_;
817     wire [7:0] _184_, _185_;
818     wire cigen_last_clk, shift_tx_clk, _179_, _180_, _194_;
819     wire [7:0] shift_tx_bit_pos;
820     wire [127:0] _510_ = shift_data;
821     assign _008_ = wb_cyc_i & wb_stb_i;
822     assign spi_tx_sel[1] = _008_ & _019_;
823     assign _019_ = wb_addr_i[4:2] == 32'd1;
824     assign shift_tx_clk = _194_ & _096_;
825     assign _088_ = ! shift_tip;
826     assign _093_ = ! _177_;
827     assign cigen_last_clk = ! _179_;
828     assign _096_ = ! cigen_last_clk;
829     assign _097_ = shift_tx_clk || _088_;
830     assign _177_ = ! ctrl[6:0];
831     assign _179_ = ! shift_cnt;
832     assign _180_ = 510 [shift_tx_bit_pos[6:0] +: 1];
833     assign _184_ = { _093_, ctrl[6:0] } - shift_cnt;
834     assign _185_ = shift_cnt - 8'h01;
835     assign _056_ = _097_ ? _180_ : shift_s_out;
836     assign shift_tx_bit_pos = ctrl[11] ? _184_ : _185_;
837     assign _194_ = ctrl[10] ? cigen_neg_edge : cigen_pos_edge;
838     always @(posedge clk) begin
839         shift_s_out <= _056_;
840     end
841 endmodule

```

(a) Code



(b) Graph

****Functionality Summary****
The Verilog design, 'shift_s_out', is a combinational circuit that takes multiple inputs and produces a single output, 'shift_s_out'. The circuit appears to be a part of a larger system, likely a **SPI (Serial Peripheral Interface) transmitter**, and is responsible for **shifting data out of a shift register**. The circuit's functionality can be broken down into several key components:

- 1. **Shift Register Control**:** The circuit takes input 'shift_cnt' and uses it to control the shift register. It also takes input 'shift_tip' to determine when to stop shifting.
- 2. **SPI Transmission**:** The circuit generates a clock signal, 'shift_tx_clk', and uses it to shift data out of the shift register. It also generates a bit position signal, 'shift_tx_bit_pos', to select the current bit being transmitted.
- 3. **Control Signals**:** The circuit takes input 'ctrl' and uses it to control the shift register and SPI transmission. It also generates several control signals, such as 'cigen_pos_edge' and 'cigen_neg_edge', which are used to control the clock signal.
- 4. **Data Shifting**:** The circuit takes input 'shift_data' and shifts it out of the shift register using the 'shift_tx_clk' signal.

****Implementation Details****
The Verilog design uses a combination of assignment statements and an always block to implement the circuit's functionality.

- * Assignment statements** are used to define the relationships between the circuit's inputs and outputs. For example, 'assign _008_ = wb_cyc_i & wb_stb_i;' defines the value of '_008_' as the logical AND of 'wb_cyc_i' and 'wb_stb_i'.
- * The always block** is used to update the 'shift_s_out' output signal on the rising edge of the 'clk' signal. The always block contains a single statement, 'shift_s_out <= _056_;', which assigns the value of '_056_' to the 'shift_s_out' output signal.
- * The design uses several intermediate signals**, such as '_008_', '_019_', and '_180_', to simplify the implementation and improve readability. These signals are used to compute the final output value, 'shift_s_out'.

Overall, the design is well-structured and easy to follow, with clear and concise assignment statements and a simple always block.

(c) Summary

Figure 7: An example for multimodal circuit

the primary signals, wire connects the internal signals, assign represents combinational logic operations, and the always block triggered by a clock signal defines the behavior of sequential registers.

B.2.2 STRUCTURAL GRAPH

Each sub-circuit HDL code is parsed into an abstract syntax tree (AST), which is then used to construct a control data flow graph, following a similar process in Fang et al. (2023). As demonstrated in Figure 7b, the nodes represent sequential registers and combinational operators (e.g., AND, ADD, EQUAL, MUX), while the wires connecting elements in the HDL code serve as the edges between these nodes.

B.2.3 FUNCTIONALITY SUMMARY

We employ GPT-4o (Achiam et al., 2023) from Open-AI to summarize both the functionality and the implementation details of each sub-circuit HDL code. An example generated summary is illustrated in Figure 7c. A sub-circuit contains only combinational logic for a single register within a single clock cycle, making it simpler for the LLM to analyze without dealing with the complex sequential state transitions of the entire circuit.

B.2.4 NETLIST GRAPH

We follow a similar widely adopted method (Wang et al., 2022) to convert netlists into the graph format. Specifically, register flip-flops (FF) and logic gates (e.g., AOI, INV, FA, AND) are treated as the nodes, and the wires connecting these gates form the edges of the graph.

B.3 SUB-CIRCUIT GENERATION ALGORITHM

We convert the HDL code into sub-circuit code snippets and the circuit graph into corresponding sub-graphs, using the same sub-circuit generation method for both modalities to ensure functional alignment. The detailed splitting algorithm is provided in Algorithm 1. Specifically, for each register, we apply a breadth-first search starting from that register, backtracking through all connected combinational logic until reaching the related input/output registers. This process is highly parallelized within a design, ensuring minimal runtime.

Algorithm 1 Sub-circuit generation(s)

```

1:  $V \leftarrow \{s\}$  ▷ Set of visited nodes
2:  $Q \leftarrow \{s\}$  ▷ Queue with start node
3:  $R \leftarrow \emptyset$  ▷ Set to store registers and inputs
4: while  $Q \neq \emptyset$  do
5:    $u \leftarrow \text{dequeue } Q$  ▷ Current node
6:   for all  $v \in u.\text{outgoing}$  do
7:     if  $\text{type}(v) \in \{\text{reg}, \text{in}\}$  then
8:        $R \leftarrow R \cup \{v\}$  ▷ Add register/input to set
9:       continue ▷ Skip to next node
10:    if  $v \notin V$  then
11:       $Q \leftarrow Q \cup \{v\}$  ▷ Add unvisited node to queue
12:       $V \leftarrow V \cup \{v\}$  ▷ Mark node as visited
13:       $v.\text{setParent}(u)$  ▷ Set parent node
14: return  $R$  ▷ Return set of all registers and inputs

```

C IMPLEMENTATION OF CIRCUITFUSION

C.1 MODEL HYPERPARAMETERS

We first detail the hyperparameters for the proposed unimodal encoders: For the **Graph encoder**, we train a 7-layer graph transformer Graphormer (Ying et al., 2021) from scratch to capture the complex relationships in circuit graph semantics and structure. This encoder uses graph positional encodings supporting up to 256 in-degrees and out-degrees for centrality encoding, a maximum distance of 5 for spatial encoding, and an edge dimension of 12. It produces graph embeddings with a dimension of 768. The node features are represented by one-hot encoding of the node type, and the edge features are based on one-hot encoding of edge types, determined by the types of connected nodes. The encoder has a hidden dimension of 256 and utilizes 3 attention heads. For the **Code encoder**, we employ a frozen LLM-based general text encoder NV-Embed-V1 (Lee et al., 2024), which handles a maximum input size of 32K tokens. This model is based on Mistral-7B-v0.1 and was ranked No. 1 on the Massive Text Embedding Benchmark (MTEB) as of May 24, 2024. It generates embeddings with a dimension of 4096, which are then linearly projected to 768. As for the **Summary encoder**, it is initialized using the first 6 layers of BERT_{base} (Devlin, 2018), following the setup in (Li et al., 2021).

For the **multimodal fusion encoder**, we initialize it with the last 6 layers of BERT_{base}. The fusion encoder is equipped with cross-attention mechanisms to enable the effective fusion of embeddings generated from the three unimodal encoders.

For the **auxiliary netlist encoder**, since graph transformers struggle with large and bit-blasted netlist graphs, we instead use a 3-layer GraphSAGE (Hamilton et al., 2017) GNN with a hidden dimension of 256. It encodes the netlist sub-circuit graphs into embeddings of 768 dimensions.

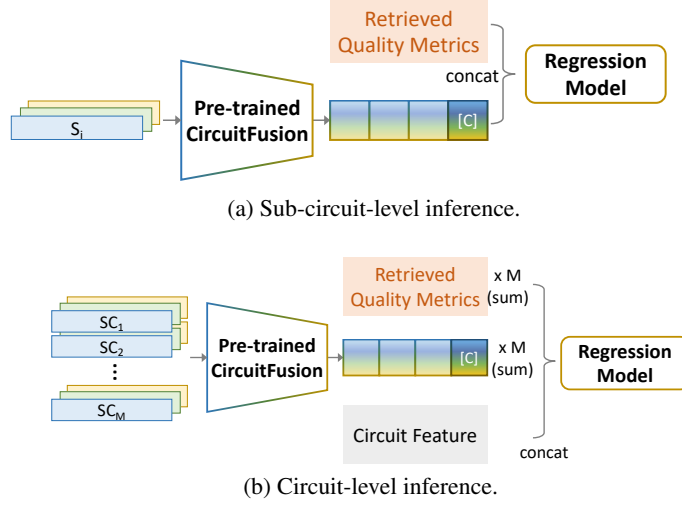


Figure 8: Retrieval-augmented inference implementation for tasks at different granularities.

C.2 SELF-SUPERVISED PRE-TRAINING TASKS IMPLEMENTATION

For masked graph modeling on both RTL and netlist sub-circuit graphs (used in Task #1 and netlist encoder), we adopt an approach inspired by GraphMAE (Hou et al., 2022). Specifically, 30% of the nodes in both the RTL and netlist graphs are randomly masked and reconstructed during each training epoch. A three-layer MLP with a hidden dimension of 256 is used to reconstruct the node types. Mean Squared Error (MSE) loss is applied to minimize the error between the original node type vectors and the reconstructed outputs, with the node types represented using one-hot encoding.

For the contrastive learning tasks (i.e., Task #1, #2, and #4), we utilize the InfoNCE loss function across all contrastive schemes. To balance the contributions of the different contrastive schemes, we assign an intra-modal weight of 1.0, while the cross-modal and implementation-aware weights are set to 0.2. The InfoNCE loss is formulated as follow:

$$NCE(E, E^+, E^-) = - \left[\log \frac{\exp(\text{sim}(E, E^+)/\tau)}{\exp(\text{sim}(E, E^+)/\tau) + \sum_{E^-} \exp(\text{sim}(E, E^-)/\tau)} \right], \quad (8)$$

where τ is the temperature scaling parameter that controls the sharpness of the similarity scores, E represents the circuit embeddings with positive samples (E^+) and negative samples (E^-). All temperature parameters are set to 0.3.

For the multimodal fusion tasks, including masked summary modeling and mixup embedding-summary matching, we adhere to the widely adopted tasks as described in (Li et al., 2021; 2022a; 2023a).

C.3 TRAINING HYPERPARAMETERS

During the pre-training phase, the four self-supervised tasks are trained simultaneously for 50 epochs, with a total training time of approximately 20 hours. We use GELU as the activation function and set the batch size to 128. For optimization, we select AdamW, known for its ability to handle large-scale data effectively. The learning rate is warmed up to $1e-4$ during the first 1000 iterations, after which it follows a cosine decay schedule, gradually reducing to $1e-5$. This schedule ensures smooth convergence while avoiding abrupt gradient updates that could destabilize the training process.

In the fine-tuning phase, the pre-trained CircuitFusion model is frozen to preserve the learned representations, and lightweight models are applied to adapt to specific downstream tasks. To complement the learned sub-circuit representations, we integrate design-level features, such as the number of different operator types, to capture the overall design scale. Specifically, we explore various lightweight models, including additional MLP layers, GNN layers, and tree-based models like XGBoost. XGBoost consistently delivers the best performance due to its capability to efficiently handle the concatenation of sub-circuit embeddings with design-level features, treating them as tabular data.

Fine-tuning requires only few computational resources, with each downstream task taking only a few minutes to complete. This rapid process facilitates quick adaptation to new tasks, ensuring that CircuitFusion is deployed efficiently across a wide range of design quality prediction applications.

C.4 RETRIEVAL-BASED INFERENCE IMPLEMENTATION

Our proposed retrieval-augmented inference process for CircuitFusion is illustrated in Figure 4. It is designed for two types of downstream tasks: sub-circuit-level and circuit-level.

Sub-circuit-level inference: For each sub-circuit, the pre-trained CircuitFusion encoder generates the corresponding multimodal embedding. We employ a retrieval process to fetch the most functionally similar sub-circuits from a vectorstore that contains previously seen circuits. These retrieved sub-circuits provide their design quality metrics, which are directly concatenated with the embedding generated by CircuitFusion. The concatenated feature vector is then fed into a regression model to predict the final design quality metric for the sub-circuit.

Circuit-level inference: At the circuit-level, the entire design is composed of multiple sub-circuits. Each sub-circuit is individually encoded by the CircuitFusion encoder, producing embeddings. Similar to the sub-circuit-level inference, we retrieve quality metrics for each sub-circuit from the vectorstore. The embeddings and retrieved quality metrics for all sub-circuits are added to generate a comprehensive circuit-level feature vector. We also concatenate this with design-level features (e.g., operator counts) to reflect the overall scale of the design. The combined feature vector is then fed into a regression model to predict circuit-level design quality metrics.

D EXPERIMENTAL SETTINGS

Our CircuitEncoder is implemented in Python, utilizing Pytorch and DGL (Wang, 2019) for self-supervised pre-training and model implementation. Experiments are conducted on a server equipped with a 2.9 GHz Intel Xeon(R) Platinum 8375C CPU and 512 GB RAM, with four NVIDIA A4000 GPUs for model pre-training.

E MORE EXPERIMENTAL RESULTS

E.1 BASELINE MODELS (EXTENDED)

We summarize the baseline model size compared with CircuitFusion in Table 4.

Table 4: Pre-trained baseline model statistics.

Model	Model Size	Embedding Dim.	Max Token	Training Data Source
NV-embed-V1	7B	4096	32768	Various text
UnixCoder	125M	768	1024	Software code
Code T5+ Encoder	110M	768	1024	Software code
CodeSage	1.3B	768	1024	Software code
CircuitFusion	500M (+7B frozen)	768	32768	Hardware circuit

E.2 ZERO-SHOT AND FEW-SHOT INFERENCE (EXTENDED)

Tables 5 to 9 illustrate the performance of CircuitFusion compared to SOTA baselines for zero-shot and few-shot learning on five design quality prediction tasks. The baseline method is selected as the top-performing model from all baselines in Table 2. The x-axis represents the fraction of training data used, ranging from zero-shot (0%) to full-shot (100%), while the y-axis shows the MAPE. These results demonstrate CircuitFusion’s effectiveness in both zero-shot and few-shot settings, making it a versatile and reliable model for early-stage design quality prediction tasks, where access to large datasets is often restricted.

Zero-shot. Only CircuitFusion supports this zero-shot capability due to our innovative retrieval-augmented method. While the baselines do not provide predictions in the zero-shot setting, Circuit-

Fusion achieves reasonable prediction accuracy without any training data, demonstrating its unique advantage.

Few-shot. CircuitFusion is particularly effective when training data is limited, which is crucial given the data availability challenges in hardware circuit design. As more training data is introduced (from 1/8 to full-shot), CircuitFusion consistently outperforms the baselines across all tasks, showing steeper performance improvements. It achieves lower MAPEs in nearly all cases, highlighting its superior ability to generalize and learn with minimal data.

Table 5: Few-shot results (MAPE) on slack prediction (Sub-Circuit-level).

Task: Slack	100%	50%	25%	13%	0%
SOTA (RTL-Timer)	15%	16%	19%	30%	N/A%
CircuitFusion	12%	14%	16%	19%	21%

Table 6: Few-shot results (MAPE) on WNS prediction (Circuit-level).

Task: WNS	100%	50%	25%	13%	0%
SOTA (RTL-Timer)	16%	29%	36%	43%	N/A
CircuitFusion	11%	17%	18%	25%	27%

Table 7: Few-shot results (MAPE) on TNS prediction (Circuit-level).

Task: TNS	100%	50%	25%	13%	0%
SOTA (RTL-Timer)	25%	35%	49%	74%	N/A
CircuitFusion	15%	24%	41%	52%	59%

Table 8: Few-shot results (MAPE) on Power prediction (Circuit-level).

Task: Power	100%	50%	25%	13%	0%
SOTA (MasterRTL)	26%	37%	46%	55%	N/A
CircuitFusion	13%	34%	43%	54%	62%

Table 9: Few-shot results (MAPE) on Area prediction (Circuit-level).

Task: Area	100%	50%	25%	13%	0%
SOTA (MasterRTL)	16%	33%	46%	56%	N/A
CircuitFusion	11%	30%	45%	51%	58%

E.3 ABLATION STUDY

Effectiveness of proposed strategies. Figure 9 shows our ablation study by removing key components employed in CircuitFusion strategies. Removing the sub-circuit generation severely limits CircuitFusion’s ability to handle large-scale circuits, leading to the most significant error increases across all tasks. Without this splitting, the model struggles to capture fine-grained circuit details, which is essential for tasks like slack prediction that require sub-circuit-level embeddings. We further assess the impact of each pre-training objective by selectively removing them. In every case, this leads to a clear rise in MAPE, indicating the importance of each pre-training task in enhancing both structural and semantic circuit understanding. Excluding retrieval-augmented inference results in a substantial increase in MAPE across all tasks. This highlights the significant role retrieval plays in enhancing fine-tuning performance by utilizing functionally similar existing circuits as references.

Impact of circuit modality. In addition to the ablation study that evaluates the use of each modality individually in Figure 1, we also conduct an extended study on the selective removal of each modality. This study aims to further quantify the contribution of each modality (i.e., code, graph, and

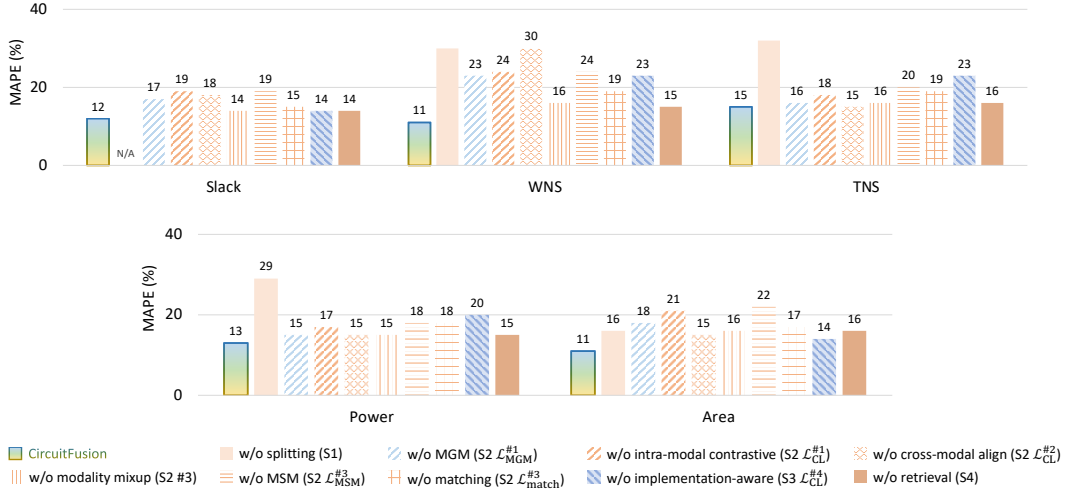


Figure 9: Ablation study on the effectiveness of proposed strategies.

summary) to the model’s overall performance. Specifically, when either the hardware code or graph modality is removed, there is a significant rise in prediction error across all tasks, highlighting their critical role in capturing both the structural and functional details of circuits. The graph modality, in particular, contributes more, as it contains rich structural information essential for circuit representation. These results demonstrate the necessity of leveraging modality fusion to fully capture the diverse characteristics of circuits.

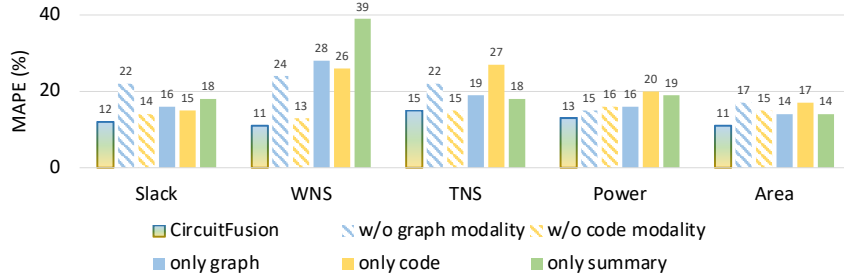


Figure 10: Ablation study on the impact of circuit modalities.

E.4 APPLYING PROPOSED STRATEGIES TO BASELINE ENCODERS

As shown in Table 10, applying the sub-circuit generation (S1) and retrieval-augmented inference (S4) strategies to other pre-trained baseline encoders significantly boosts their performance across all tasks. By encoding sub-circuits instead of the entire circuit, all baseline methods are now able to handle the fine-grained slack prediction task, which they originally could not support.

For example, the LLM-based encoder NV-Embed-v1, despite its ability to process 32k tokens, struggles to encode entire circuit code sequences. When enhanced with S1 and S4, it achieves a notable reduction in MAPE for WNS (from 26% to 17%), TNS (from 55% to 27%), power (from 44% to 20%), and area (from 24% to 17%). Similarly, other software code encoders, such as CodeSage, CodeT5+ Encoder, and UnixCoder, also benefit significantly from these strategies. This shows that S1 and S4 not only improve fine-tuning accuracy but also enhance generalization across various design quality prediction tasks. Despite these improvements, CircuitFusion still outperforms all baselines, underscoring the effectiveness of its hardware-specific pre-training strategies.

Table 10: Evaluation results when applying strategy S1 and S4 to other pre-trained encoders.

Method		Slack		WNS		TNS		Power		Area	
		R	MAPE	R	MAPE	R	MAPE	R	MAPE	R	MAPE
NV-Embed-v1	ori	N/A		0.49	26%	0.97	55%	0.85	44%	0.86	24%
	w/ S1&4	0.85	15%	0.81	17%	0.95	27%	0.99	20%	0.97	17%
CodeSage	ori	N/A		0.23	21%	0.86	45%	0.8	38%	0.77	41%
	w/ S1&4	0.84	14%	0.9	25%	0.95	24%	0.96	18%	0.96	17%
CodeT5+ Encoder	ori	N/A		0.55	30%	0.63	43%	0.49	46%	0.45	39%
	w/ S1&4	0.83	14%	0.8	21%	0.94	24%	0.95	19%	0.93	21%
UnixCoder	ori	N/A		0.46	21%	0.95	44%	0.83	29%	0.85	26%
	w/ S1&4	0.84	14%	0.83	20%	0.96	22%	0.96	18%	0.96	16%
CircuitFusion		0.87	12%	0.91	11%	0.99	15%	0.99	13%	0.99	11%