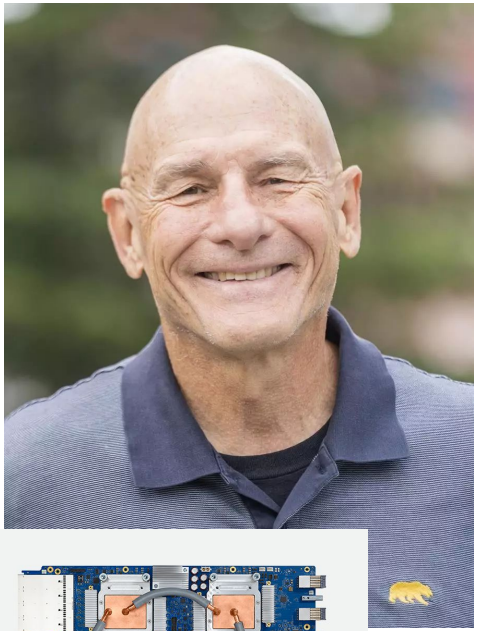
Designing Hardware for Machine Learning and Using Machine Learning to Design Hardware

John Wawrzynek

johnw@berkeley.edu
University of California, Berkeley

Machine Learning has been Great for Computer Architects!

- With the slowing of Moore's Law and the end of Dennard Scaling, architects have turned to "accelerators" and purpose built processors as a way to continue to scale performance, energy efficiency, and cost.
- For ML kernels, relatively easy to achieve high efficiency (compared to general purpose code) -
  - Simpler control flow.
  - Embarrassing parallel (loads of data-level parallelism),
  - lower precision requirements.
- Highly impactful!
- No end in sight!



Cloud TPU v3 (45 TFLOP/s)

Outline

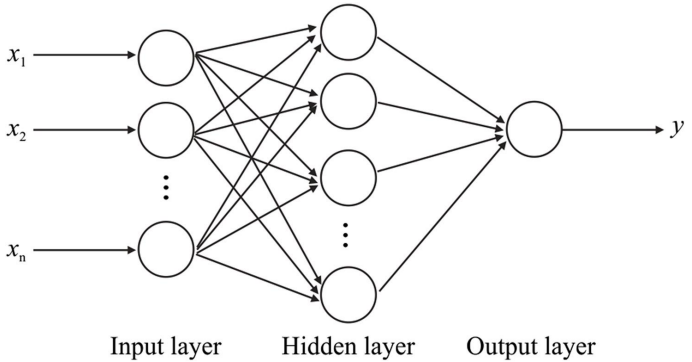
Past

Present

Future

Work at Berkeley - Early 1990's

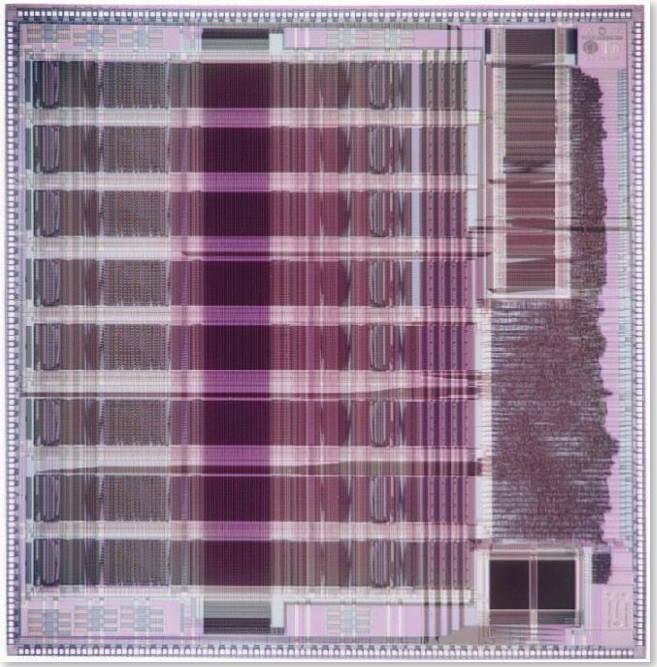
- Community wide keen interest in *parallel processing*
  - *How to we find parallelism in problems and exploit in hardware?*
- ICSI was successful at using ANN's trained with back-propagation for front-end signal processing of speech signals for speech recognition tasks. MLPs not DNNs!
- Training was taking months on CPUs.



- Our pitch: connectionist models (neural networks) could be a general model for many computations and are naturally parallel.
- Got funded from the ONR to design and build a "connectionist network supercomputer (CNS)".
- Quickly realized that training and inference is dominated by multiply/add operations and these vectorize => designed a vector processor.

T0 Vector Microprocessor (1995)

- World's first single chip vector processor
- MIPS CPU + vector lanes
- Three graduate students, 1 year
- 15X the performance of workstation

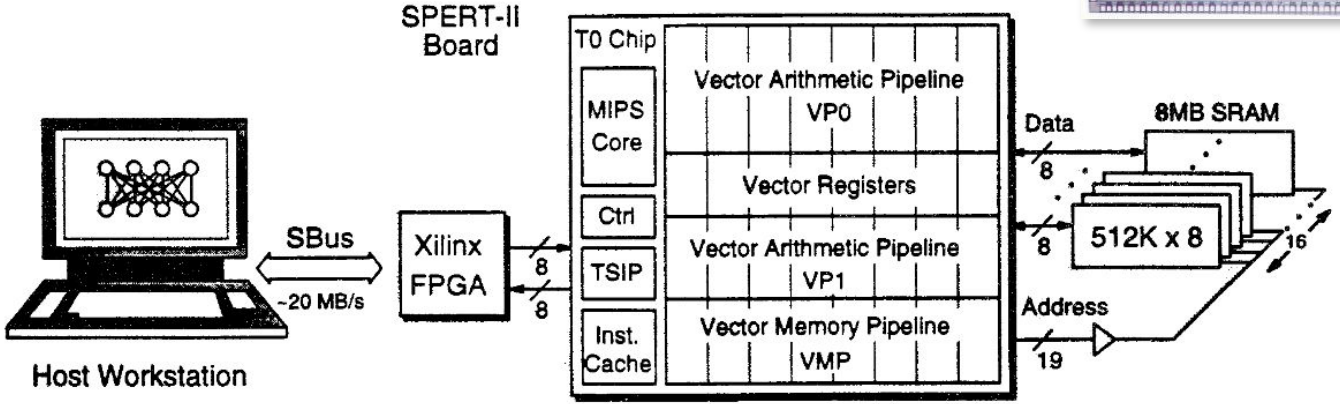


SPERT-II: A Vector Microprocessor System and its Application to Large Problems in Backpropagation Training

John Wawrzynek, Krste Asanović, & Brian Kingsbury
University of California at Berkeley
Department of Electrical Engineering and Computer Sciences
Berkeley, CA 94720-1776

James Beck, David Johnson, & Nelson Morgan
International Computer Science Institute
1947 Center Street, Suite 600
Berkeley, CA 94704-1105

Proceedings of MicroNeuro '96



Libraries for Speech researchers

System lived on for a decade!

Lessons from the 1990's

1. NN training/inference dominated by multiply/add operations
   - Full precision rarely needed
   - Data-level parallelism (vectors/matrices)

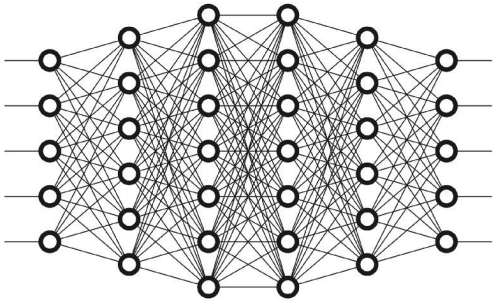
Obviously lives on today with TPUs, etc.

2. ANN's good general computation model
   - Wide range of function approximation, regression, classification, etc., useful tool in optimization.

3. Software is key to adoption

GPUs with Cuda/openCL, now TensorFlow/PyTorch

Learning Model Capabilities Scaled Directly with Hardware Advances



- Late 2000's - renewed interest in NNs, now **deep**
- Driven by availability of high-performance hardware (GPUs)
- ML models and HW development fundamentally linked:
  - Success in LLMs tied directly to massive hardware compute capability (even more important than algorithm details?)

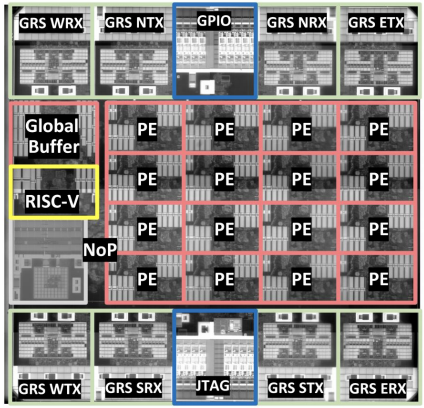
Example: LLMs
GPT-4 trained on ~25,000 Nvidia A100 GPUs for 90-100 days,
~1.8 trillion parameters across 120 layers (~13T tokens in training)
[https://archive.md/2RG8X]

- How can we continue to scale HW performance (efficiency) to the benefit of ML?
  - *Scalable HW architectures + HW/Algorithm co-design*

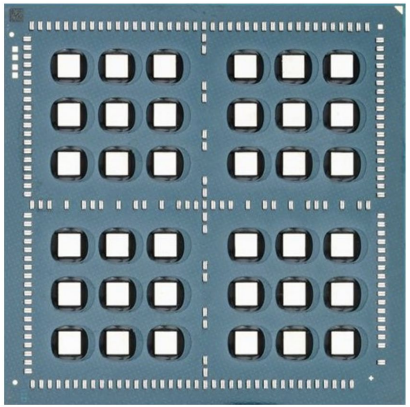
Increasing Number of Parallel Resources

Many PEs with Network on Chip/Package (NoC/NoP)

NoC/NoP Chip



(a) Simba chiplet

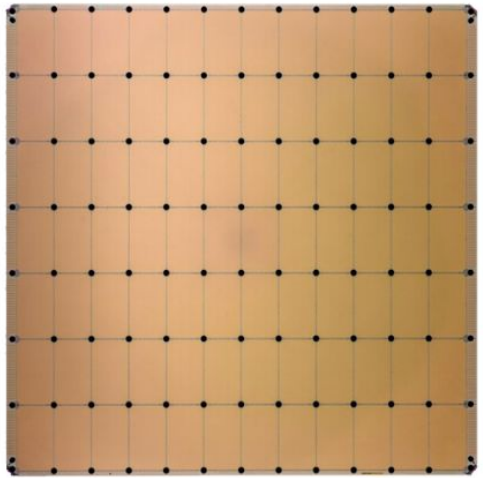


(b) Simba package

Simba
16PEs x 36 Chiplets

Sophia Shao NVIDIA/UC Berkeley

Wafer-scale Chip



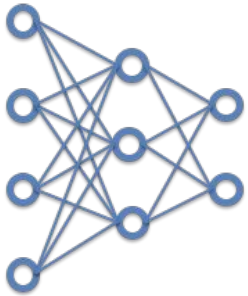
Cerebras WSE-3
4 Trillion Transistors
46,225 mm² Silicon

Largest GPU
80 BillionTransistors
814 mm² Silicon

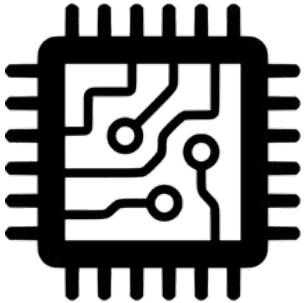
Cerebras
84 Interconnected Chips

Scheduling a constant challenge:

Particularly for multi-core architectures. How to partition and schedule execution to efficiently use parallel resources.



Scheduling



- Algorithm

Problem instances are huge (large amount of state, large number of operations)

- Hardware

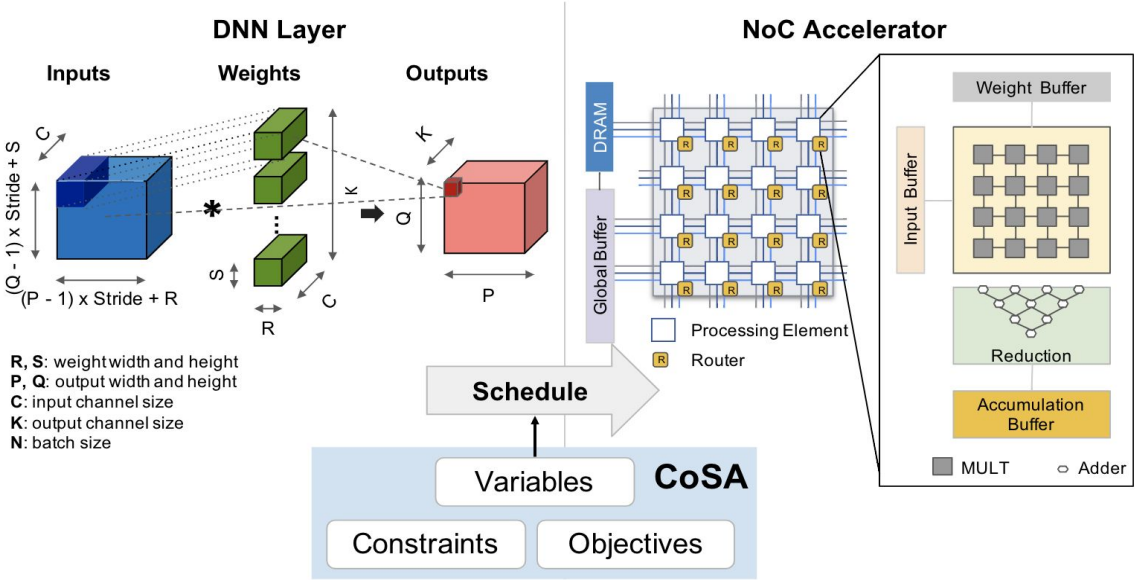
Relatively small amount of fast hardware resources (memory, computational units)

Perhaps the most important part of support software.

CoSA: Scheduling by Constrained Optimization

patial Accelerators [21'ISCA]

Jenny Huang PhD Dissertation



Scheduling Decisions:

1. Loop tiling
2. Loop permutation
3. Spatial mapping

Objective is to minimize latency and energy

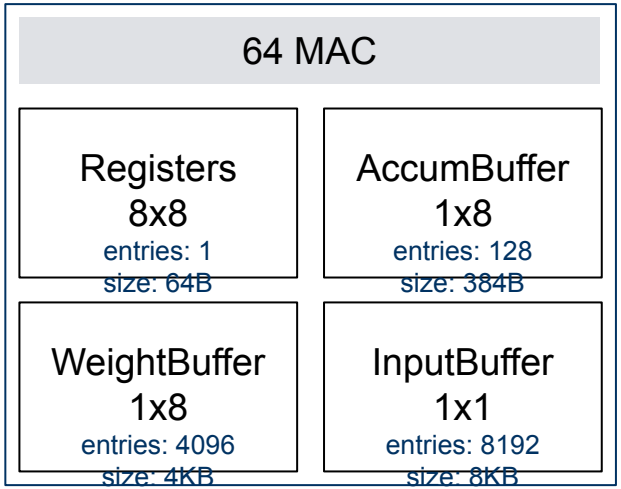
Mixed Integer Programming (MIP) for scheduling DNN on NoC accelerator with multi-level memory hierarchies

Three operation-level scheduling decisions

- **Inputs Constraints:**

Problem: ----- 7 nested loops
R=3, S=3, P=28, Q=28, C=128, K=128, N=1

Architecture: ----- 5 levels of memory



- **Output Schedule:**

DRAM [Weights:147456 Inputs:115200 Outputs:100352]

| for P in [0:4)
| for S in [0:3)
| for C in [0:16) (Spatial-X)
InputBuffer [Inputs:2016]

| for N in [0:1)
| for R in [0:3) (Spatial-X)
WeightBuffer [Weights:1024]

| for Q in [0:28)
| for P in [0:7)
AccumulationBuffer [Outputs:128]

| for K in [0:128)

1. Loop Permutation
2. Spatial Mapping
Temporal Mapping
3. Tiling Factors

State-of-the-art DNN accelerator schedulers

Scheduler	Search Algorithm
Brute-force Approaches:	
Timeloop [57]	Brute-force & Random
dMazeRunner [28]	Brute-force
Triton [75]	Brute-force over powers of two
Interstellar [81]	Brute-force
Marvel [17]	Decoupled Brute-force
Feedback-based Approaches:	
AutoTVM [19]	ML-based Iteration
Halide [65]	Beamsearch [4], OpenTuner [9], [52]
FlexFlow [42]	MCMC
Gamma [45]	Genetic Algorithm
Constrained Optimization Approaches:	
Polly+Pluto [15], [16], [35]	Polyhedral Transformations
Tensor Comprehension [77]	
Tiramisu [11]	
CoSA	**Mixed Integer Programming (MIP)**

1. Expensive and time-consuming
2. Sample invalid space
3. Hard to generalize

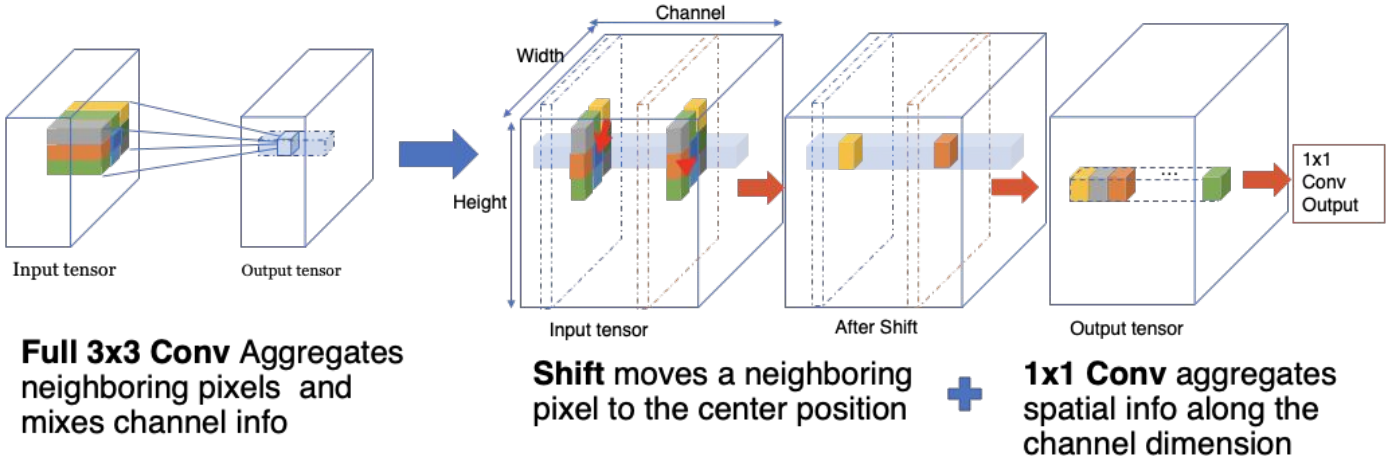
Unable to determine tiling factor sizes

One-pass solution

2x speedup compared to the state-of-the-art work with **116x** shorter time-to-solution

Hardware-Friendly Algorithm Design

Synetgy: Image Classification without 3x3 Convolution [FPGA'19]

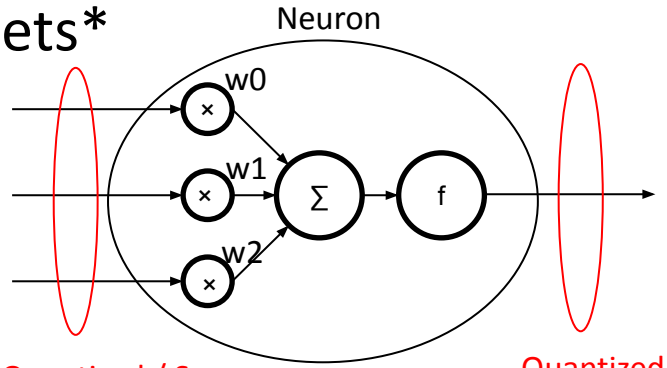


- 3x3 Conv → Shift and 1x1 Conv
- Dataflow accelerator on embedded FPGA

equal top-1 accuracy, **11.6x** higher frame-rate, **6.3x** better power efficiency, on ImageNet classification task

Synthesis of Efficient Neural Networks on FPGAs

LogicNets*



Truth table

In0	In1	In2	In3	In4	in5	Out0	Out1
0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	1
0	0	0	0	1	0	0	0
0	0	0	0	1	1	1	1
0	0	0	1	0	0	1	0

Use Vivado LS to generate FPGA mapping

Benchmark	Accuracy	Throughput	Latency	LUT
JSC	-3.2%	1.9x	3.8x	2.3x
NID	1.2%	9.2x	1.9x	3.2x

Improvement (over state-of-art)

- Jet Substructure Classification (JSC)
  - Particle physics experiments at Large Hadron Collider at CERN, 16 inputs (FP32), 5 classes
- Network Intrusion Detection (NID)
  - Detecting malicious network packets, 49 inputs (binary), 9 classes

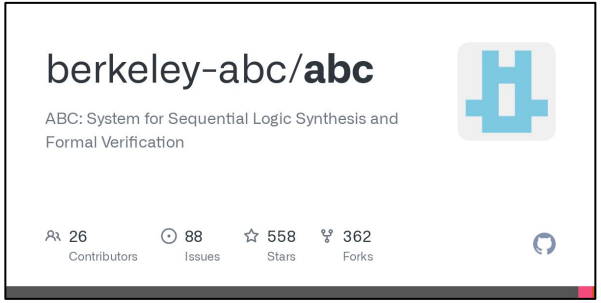
[*] Y. Umuroglu et al., “Logicnets: Co-designed neural networks and circuits for extreme-throughput applications,” in Proceedings of FPL, 2020, pp. 291–297.

Berkeley Wireless Research Center

LogicNets at Berkeley

- **Developed a new logic synthesis algorithm for LogicNets**
- The truth table converted from a neuron can be classified as

  Random-looking dense function with limited support
  - Looks almost random
  - Does not have a compact SOPs
  - Depend only on a few inputs (up to 16 bits)
- We developed a dedicated logic synthesis algorithm for this class of functions - based on Binary Decision Diagram (BDD) minimization - variable reordering



Alan Mishchenko

Yukio Miyasaka

Synthesis of LUT Networks for Random-Looking Dense Functions with Don't Cares—Towards Efficient FPGA Implementation of DNN

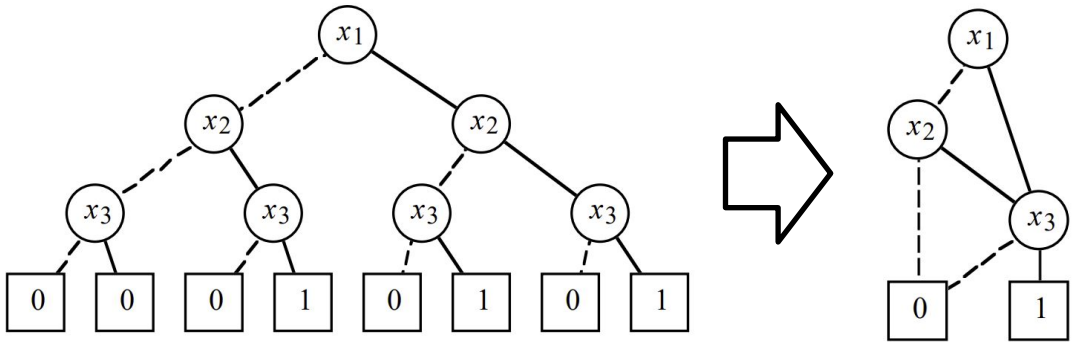
Y Miyasaka, A Mishchenko, J Wawrzynek, NJ Fraser

2024 IEEE 32nd Annual International Symposium on Field-Programmable Custom …

Overview of the optimization algorithm

- Construct a Binary Decision Diagram* (BDD) for each neuron
- Perform BDD minimization (variable reordering)
- Map BDDs into FPGA LUTs

x_1	x_2	x_3	f
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1



[*] Randal E. Bryant, “Symbolic Boolean manipulation with ordered binary-decision diagrams,” in ACM Computing Surveys, vol. 24, no. 3, pp. 293–318, 1992.

Synthesis of Efficient Neural Networks on FPGAs

Minimization using don't cares

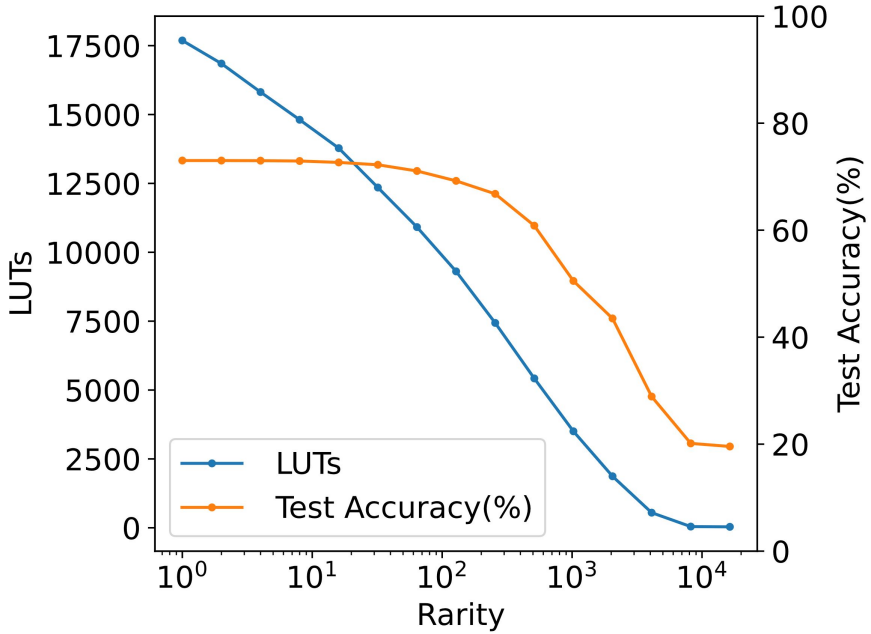
- We can further reduce the area by assigning some patterns to don't care
  - Allow the function to change during optimization
  - Trade-off area with accuracy
- To begin with, we assigned patterns that do not appear in training-set to don't care
- We also tried setting a threshold, named rarity
  - Only the patterns with higher occurrences in training-set will be cared
  - Higher rarity allows the function to change more

In0	In1	Out0	Out1
0	0	1	0
0	1	0	1
1	0	1	1
1	1	1	0

In0	In1	Out0	Out1
0	0	1	0
0	1	Φ	Φ
1	0	1	1
1	1	1	0

Synthesis of Efficient Neural Networks on FPGAs

Effect of rarity parameter on size and test accuracy for JSC benchmark



- By assigning more patterns to don’t care, we can further reduce the area
- The accuracy drops quite slowly compared to the area reduction

Results of minimization using don't cares for JSC benchmark

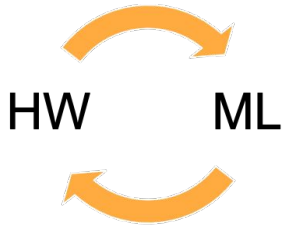
Method	Accuracy	LUT	Time
Xilinx Vivado	73.0171%	35419	2373s
w/o don't care	73.0171%	22997	144s
w/ don't care	73.0146%	17687	178s

- By assigning the patterns not in training-set to don't care,
  - With don't cares, area reduced 2x over Vivado's result
  - accuracy was degraded only 0.0025%
- Runtime was still 13x smaller than Vivado

The Future

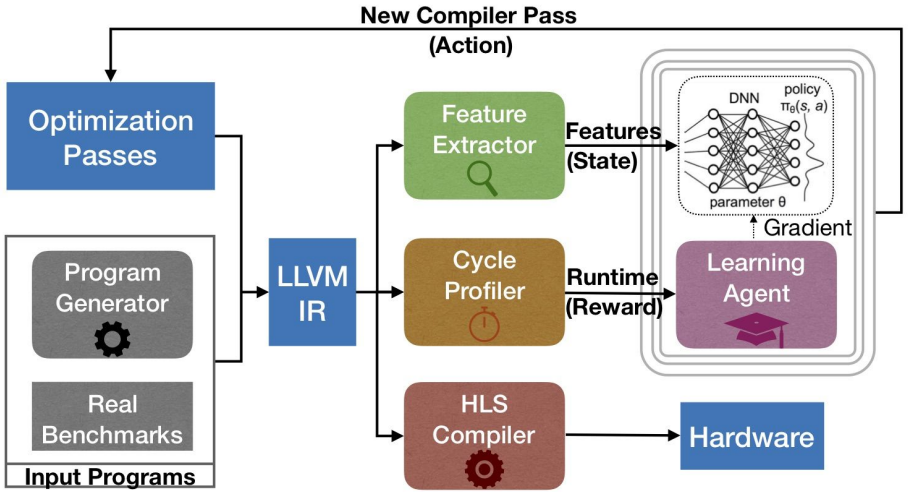
- **Advances in ML will continue to be critically dependent on advances in Hardware Design.**
  - To spur ML advances, HW design must be agile: easy, fast, cheap
    - None of these true now!
    - Chip/accelerator design is slow, expensive (years, $10-100M)
    - Consequently, we use yesterday's application benchmarks to design tomorrow's HW!
- **New work shows promise in applying ML to HW design challenges:**
  - Higher QoR: DNNs classifiers allow rapid DSE and aids verification
  - Higher human productivity: Reinforcement learning automates optimal search

<u>Virtuous Cycle:</u> Applying ML algorithms to design HW leads to better HW for accelerating ML which will lead to better ML algorithms for HW design which will lead to …



AutoPhase: Reinforcement Learning for HLS Phase-Ordering [MLSys'20]

Jenny Huang



NP-Hard

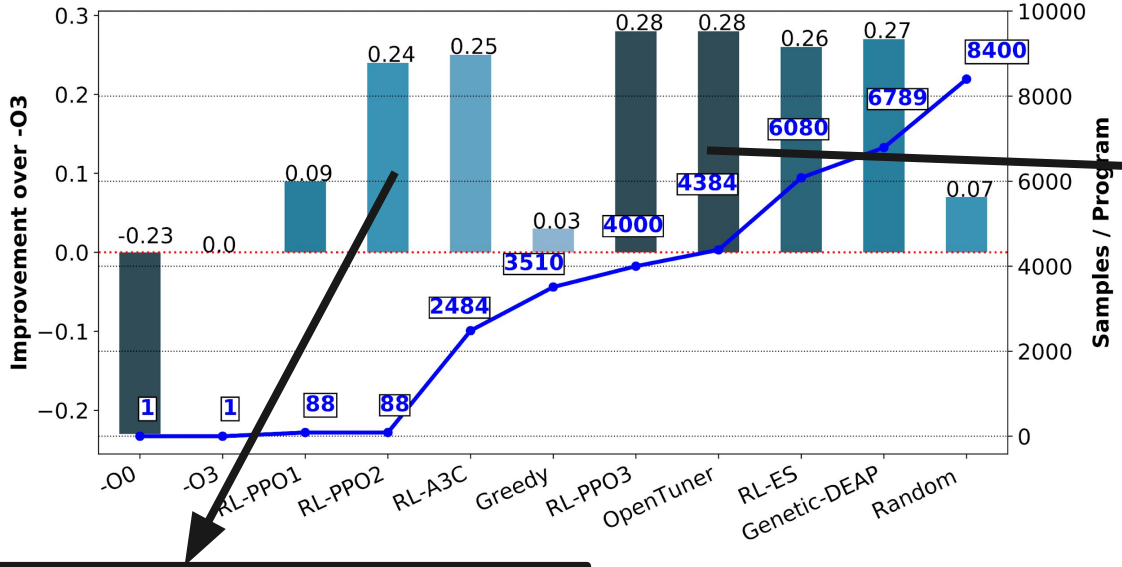
Phase-ordering:

clang program.c -flag1 -flag2 ...

We apply **deep reinforcement learning** to address the phase-ordering problem for HLS:

- Action: next optimization pass to apply
- States:
  1. Program Features
  2. Histogram of Applied Passes
- Reward: cycles before a pass is applied − cycles after a pass is applied

AutoPhase: Reinforcement Learning for HLS Phase-Ordering [MLSys'20]



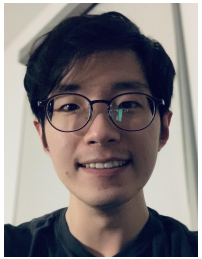
Uses **Program Features** and **Histogram of Applied Passes**

- Uses only **Histogram of Applied Passes**
- Compiles and runs **once** per trajectory

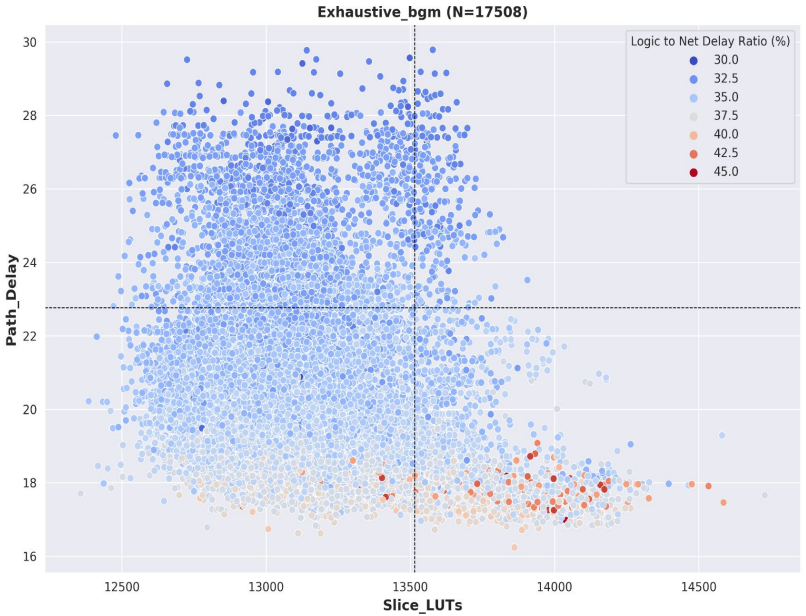
- **Fewer** samples required
- **28%** improvement over -O3

Logic Synthesis QoR optimization (‘22)

- Logic Synthesis results can dictate the viability of final QoR optimization
  - Under-optimized synthesized netlist → sub-optimal post-PnR QoR
- However, diverse options for Boolean logic optimizations (e.g. passes in open-source synthesis tool ABC)
  - Search space of possible “recipes” grows exponentially w.r.t. length of sequence of passes
- Users traditionally rely on ad hoc heuristics or standard (built in) synthesis scripts:
  - This work studies a ML data-driven approach



Josh Kang



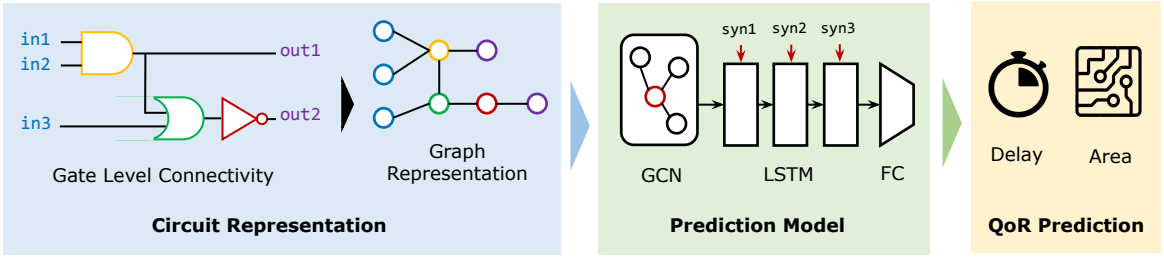
Preliminary DSE (FPGA Mapping)

Dotted: Default Yosys-ABC Recipe

Best Recipe is Circuit Dependent!
How to find the best recipe?

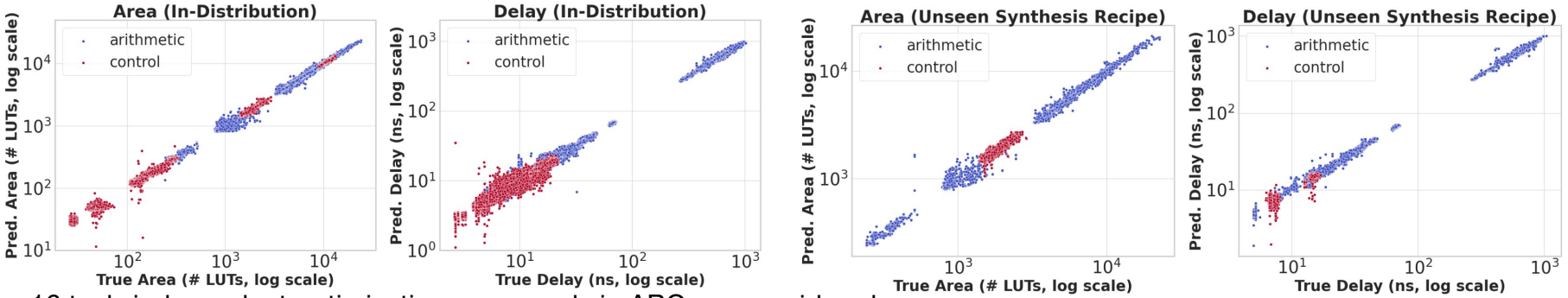
Berkeley Wireless Research Center

GCN-LSTM Synthesis QoR Prediction



QoR	MAPE (%)	Correlation
Delay	2.674 (%)	0.997
Area (# LUTs)	3.978	0.998

End-to-end prediction model architecture:
4-layer GCN, 4-layer LSTM with all hidden dimensions fixed to 64

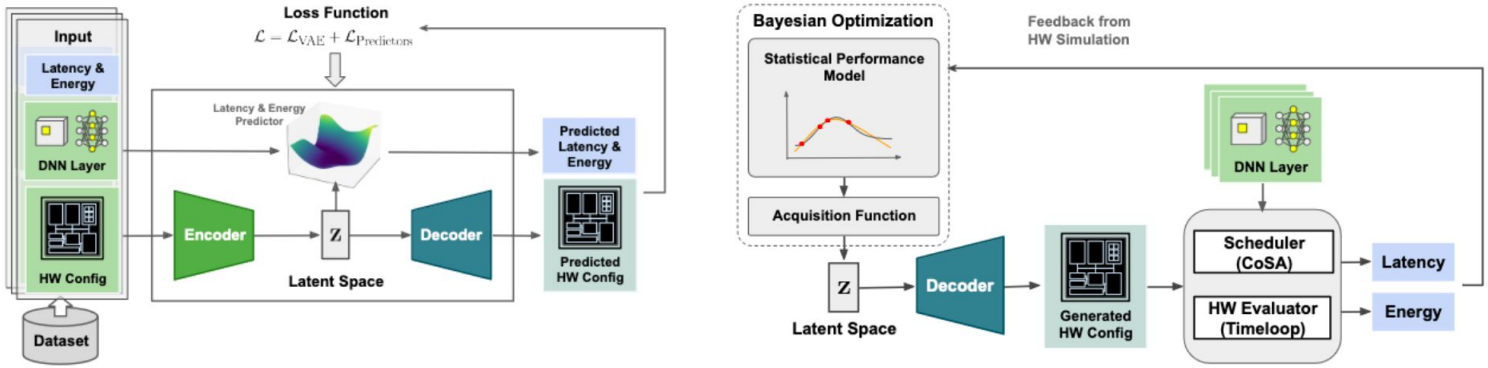


16 tech-independent optimization commands in ABC are considered.

Supervised dataset with ground-truth post-synthesis QoR labels (from Vivado Design Analysis)

Learning A Continuous and Reconstructible Latent Space for Hardware Accelerator Design (VAESA)

2022 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)

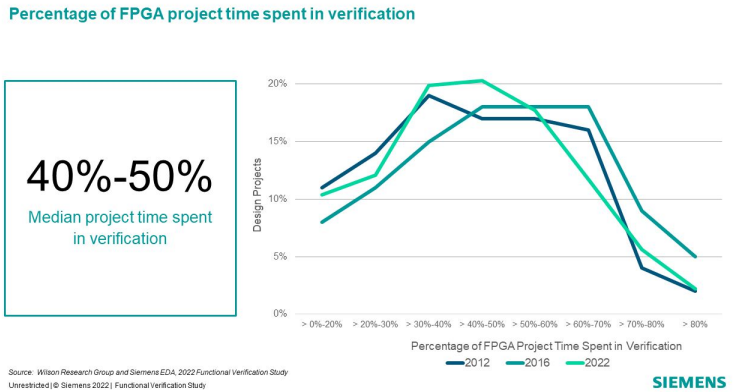


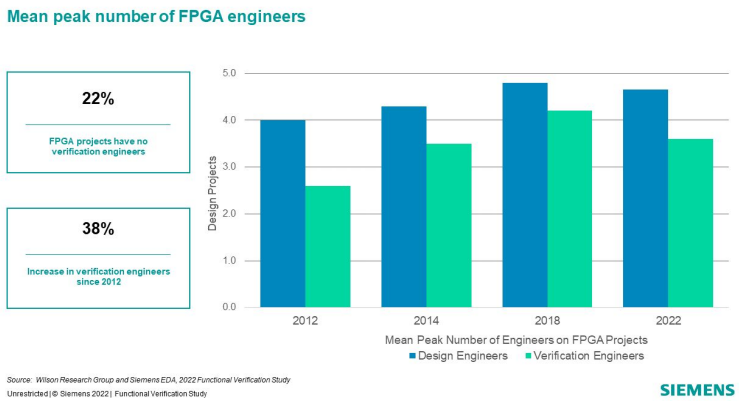
- HW design space exploration (DSE) exponentially large in design parameters & discontinuous
- Design challenge: Design Simba-like (multicore) architectures for DNNs optimizing for latency and energy
  - parameters such as # of PEs, weight and input buffer sizes, … 3.6 x 10^{17} configurations!
- We use a *variational autoencoder (VAE)* to learn a compressed and continuous representation (latent) of the design space - new designs can be generated from the latent space
- Eases search (Bayesian optimization, and gradient-based search)
- Demonstrated on AlexNet, ResNet-50, ResNeXt-50, Deep Bench, …
- Significantly improves optimization results versus searches in original design space (5%) and 6.8X better sample efficiency

Efficient and Scalable RTL Verification

Demands for efficient and / scalable RTL verification

- Significant project time, compute, and human resources dedicated to *verification* in industrial chip design development
- Verification closure requires:
  - Formal Property Verification: overcoming the state-space explosion problem in assertion-based verification
    → *Learned Formal Proof Strengthening*
  - Constrained Random Verification: efficient generation of constraint-satisfying solutions
    → *GPU-Enabled, High-Throughput SAT Solution Sampling*

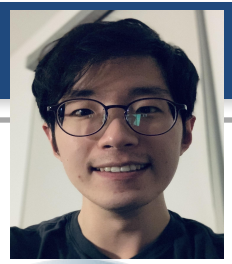




https://blogs.sw.siemens.com/verificationhorizons/2022/10/30/part-3-the-2022-wilson-research-group-functional-verification-study/

Berkeley Wireless Research Center

Learned Formal Proof Strengthening (ICCAD'23)

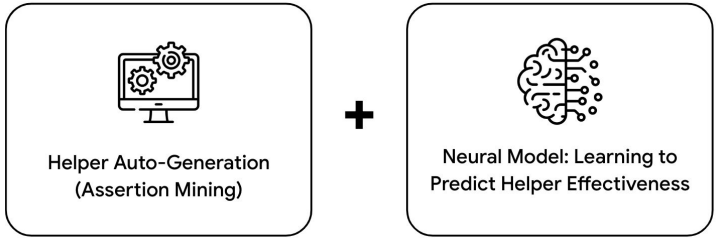


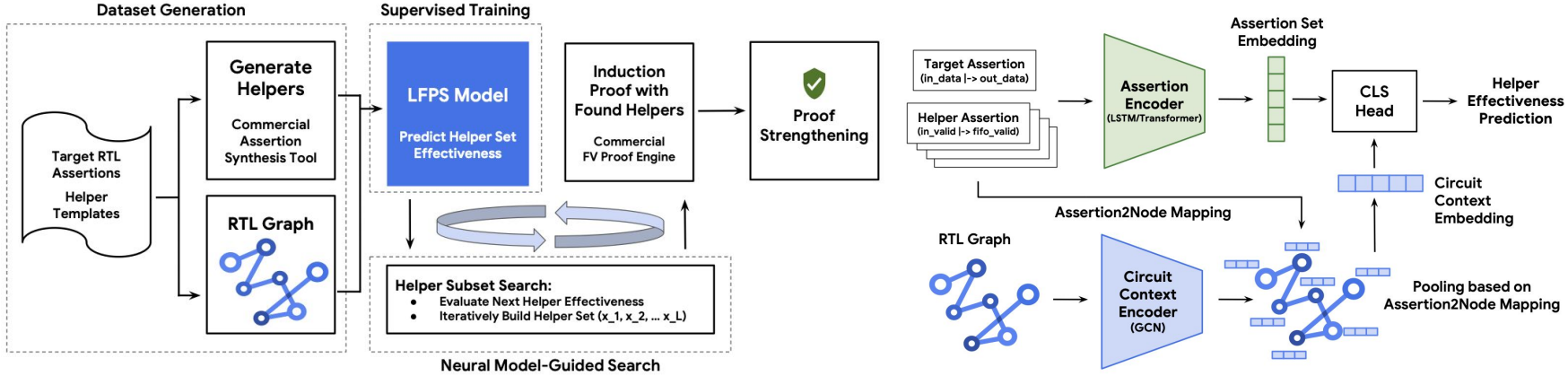
Josh Kang

IEEE/ACM 2023 INTERNATIONAL CONFERENCE ON COMPUTER-AIDED DESIGN ICCAD 42nd Edition

LFPS (ICCAD'23)

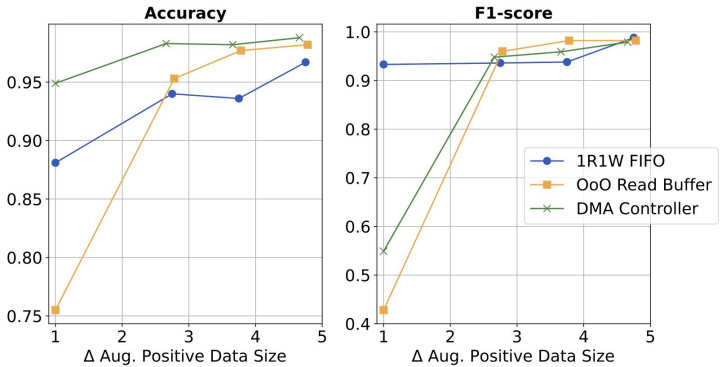
- Neural model to predict which subset of tool-generated helper invariants should be used as assumptions in k-induction proofs for assertion-based formal verification
- 98.2% prediction accuracy; 98.3% F-1
- Data augmentation improves prediction by up to 23.2%





Learned Formal Proof Strengthening (ICCAD'23)

Model	1R1W FIFO			OoO-Read Buffer			DMA Controller		
	Accuracy	Precision	Recall	Accuracy	Precision	Recall	Accuracy	Precision	Recall
MLP AE	0.698	0.711	0.804	0.769	0.764	0.967	0.925	0.667	0.053
LSTM AE	0.941	0.929	0.971	**0.982**	0.987	0.987	0.978	0.747	0.966
Transformer AE	0.946	0.957	0.948	**0.982**	0.984	**0.991**	0.987	0.969	**0.983**
Full LFPS Model (GCN CCE + Transformer AE)	**0.967**	**0.985**	**0.977**	**0.982**	**0.986**	0.990	**0.988**	**0.978**	0.981



Search Method	k	pass@k		
		1R1W FIFO	OoO-Read Buffer	DMA Controller
Random Search	1	0.24 ± 0.21	0.35 ± 0.21	0.24 ± 0.16
	3	0.47 ± 0.23	0.64 ± 0.15	0.39 ± 0.17
	5	0.55 ± 0.23	0.79 ± 0.15	**0.41** ± 0.15
Transformer AE-Only LFPS-Guided Beam Search	1	0.60	0.60	0.00
	3	0.60	0.60	0.00
	5	0.60	0.60	0.00
GCN CCE + Transformer AE LFPS-Guided Beam Search	1	0.60	**0.80**	0.20
	3	0.60	**0.80**	0.20
	5	**0.80**	**0.80**	0.20

Berkeley Wireless Research Center

GPU-Enabled SAT Solution Sampling

Differential Modeling of SAT problems

- Boolean Satisfiability critical algorithm in hardware verification (and synthesis).
  - SAT solution sampling used in Constrained Random Verification (CRV) of hardware
- Our Approach:
  - Relaxation of assignments (0 or 1) to real value between 0 and 1

$$\mathbf{x} = (x_1, x_2, ..., x_n) \in \mathbb{R}^n \qquad \tilde{\mathbf{x}} = (x_1, x_2, ..., x_n) \in \{0, 1\}^n$$

  - Model as optimization problem: convex with L2 loss. Implies guaranteed convergence with gradient descent.
  - Similar to *back-propagation* algorithm
  - "NN model" of logic circuits
- Leverage ***GPUs with standard libraries, batch processing***

ASP-DAC'25

DEMOTIC: A Differentiable Sampler for Multi-Level Digital Circuits

Arash Ardakani, Minwoo Kang, Kevin He, Qijing Huang, Vighnesh Iyer, Suhong Moon, John Wawrzynek

On ISCAS-85 benchmark suite. DEMOTIC outperforms the state-of-the-art sampler by more than 100x in most cases.

CircuitSAT Instance	# Inputs	# Outputs	# Logic Gates	# Variables (CNF)	# Clauses (CNF)	DEMOTIC	UNIGEN3	CMSGEN	DIFFSAMPLER
c17	5	2	6	25	19	850	813	**162, 707**	116,968
c432	36	7	160	539	516	**2, 054, 518**	1.5	10, 070	105
c499	41	32	202	683	717	**1, 123, 605**	1.5	5, 704	28
c880	60	26	383	1198	1115	**510, 760**	0.2	4, 379	15
c1355	41	32	546	1683	1613	**648, 736**	0.2	3, 109	0.9
c1908	33	25	880	2436	2381	**367, 720**	TO	2, 213	TO
c2670	233	140	1269	3642	3274	**323, 617**	TO	1, 385	TO
c3540	50	22	1669	4680	4611	**65, 156**	TO	1, 073	TO
c5315	178	123	2307	6994	6698	**180, 085**	TO	655	TO
c6288	32	32	2416	7280	7219	**40, 325**	TO	14	TO
c7552	207	108	3513	9971	9661	**64, 483**	TO	430	TO

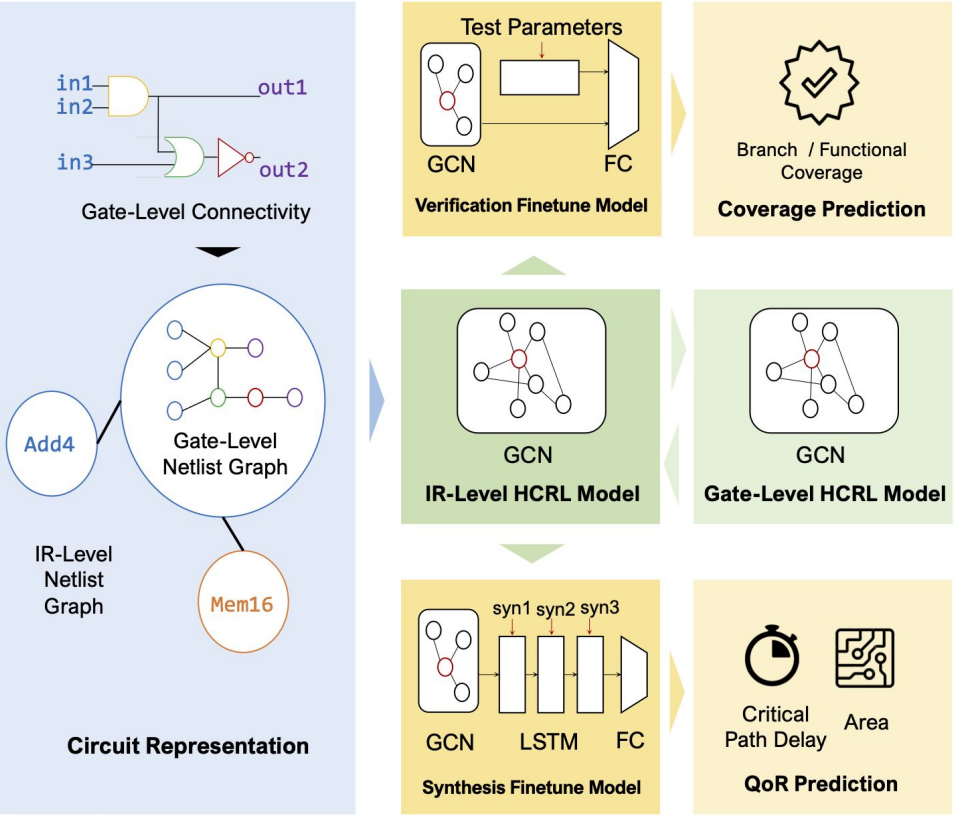
DAC'24

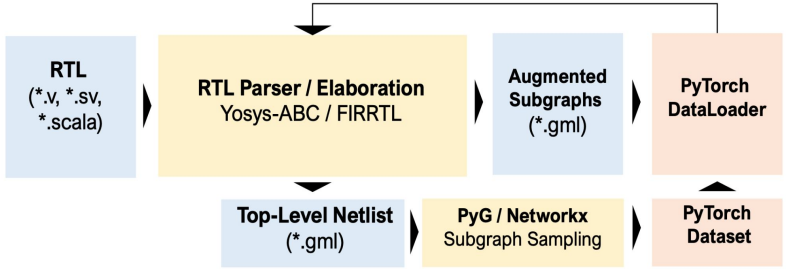
Late Breaking Results: Differential and Massively Parallel Sampling of SAT Formulas

Arash Ardakani*, Minwoo Kang*, Kevin He, Vighnesh Iyer, Suhong Moon, John Wawrzynek
University of California, Berkeley
{arash.ardakani, minwoo_kang}@berkeley.edu

Berkeley Wireless Research Center

(Ongoing) Hierarchical SSL of Digital Circuits





Self-Supervised Training for Digital Circuit Representation Learning

- Design abstraction level as inductive bias
- Scalable representation learning of larger / realistic designs, while training with low-level information
- Adequate training data remains a challenge!

Chip Layout as Image Generation

"Chip Placement with Diffusion"
Vint Lee, Chun Deng, Leena Elzeiny, Pieter Abbeel, John Wawrzynek
Under review at NeurIPS'24, https://arxiv.org/abs/2407.12282

- From macro-cell/standard-cell netlist to layout minimizing HPWL
- Existing learning-based methods use Reinforcement learning:
  - Slow, per-netlist iteration
- We employ a Denoising Diffusion Probabilistic Model (DDPM) formulation,
- with universal guidance

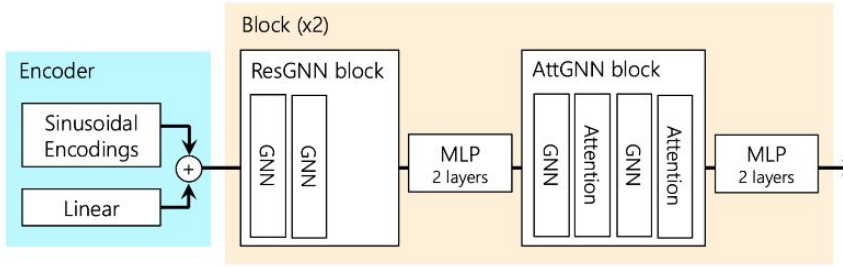
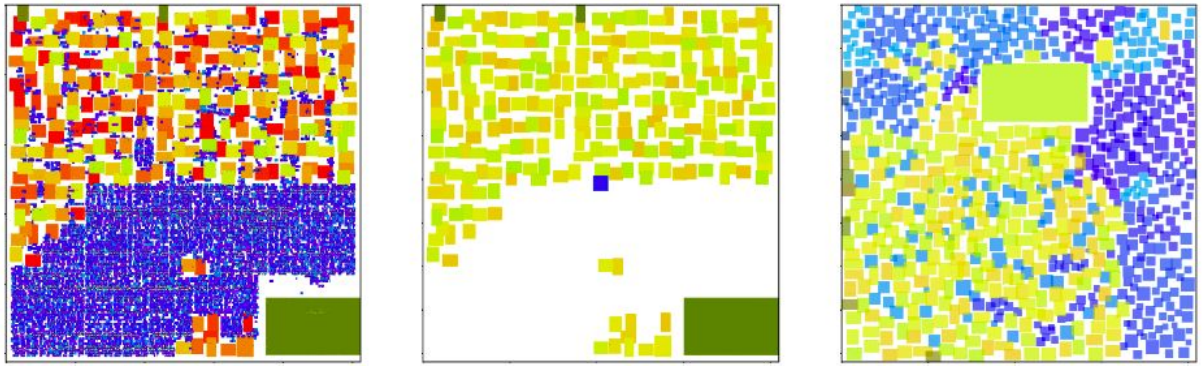
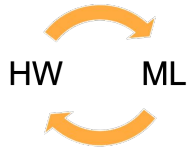


Figure 2: Diagram of our denoising model. Residual connections, edge feature inputs, nonlinearities, and normalization layers have been ommitted for clarity.



(a) Original Circuit (b) After Clustering (c) Diffusion-based Placement

The Future of ML for HW



- Emerging body of work:
  - Many solid results on ML for physical design problems (placement/routing)
  - Some in logic and high-level synthesis
  - Most build estimators of power, cost, or performance to aid in search
  - Very few positive results on “generative techniques”
    - ex: “Design an LDPC decoder with 1Gsamp/s throughput and 10mW for SK90FD”
    - LLMs good for designer/tools interaction
      - not suitable for optimization
    - “Large Circuit-Models”? “Small Circuit-Models?” “Small-circuit Models?”
      - Challenges: Training sets, and correctness guarantees, constraint satisfaction

Co-authors / Collaborators

- T0 Vector Microprocessor: *Krste Asanovic, Brian Kingsbury, James Beck, David Johnson, Nelson Morgan*
- CoSA: *Qijing Huang, Minwoo Kang, Grace Dinh, Thomas Norell, Aravind Kalaiah, James Demmel, Sophia Shao*
- Synetgy: *Yifan Yang, Qijing Huang, Bichen Wu, Tianjun Zhang, Liang Ma, Giulio Gambardella, Michaela Blott, Luciano Lavagno, Kees Vissers, Kurt Keutzer*
- LogicNets: *Yukio Miyasaka, Alan Mishchenko, Nick Fraser*
- Autophase: *Qijing Huang, Ameer Haj-Ali, William Moses, John Xiang, Ion Stoica, Krste Asanovic*
- VAESA: *Qijing Huang, Charles Hong, Mahesh Subedar, Sophia Shao*
- Learned Formal Proof Strengthening: *Minwoo Kang, Azade Nova, Eshan Singh, Geetheeka Sharron Bathini, Yuriy Viktorov*

Thanks!