Roadmap of IC Design

6.9 billion

30 billion

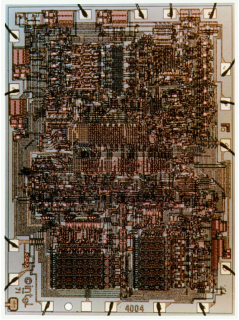
150 billion

Designed By Hand

Earliest EDA tools:
"*Berkeley VLSI Tools Tarball*"

10^3

"Quine-McCluskey", 1956



2,300

"Espresso, SIS, Magic"

1980s

1990s

10^5

Birth of Commercial EDA

Cadence, Synopsys, Mentor, etc.

Growing

"ABC, Yosys, TDS, VPR, LegUp"
"Xilinx, Intel(Altera)"

$10^{7\sim9}$

2000s

Today

10^9

A primary barrier of hardware innovation

~~Automation~~

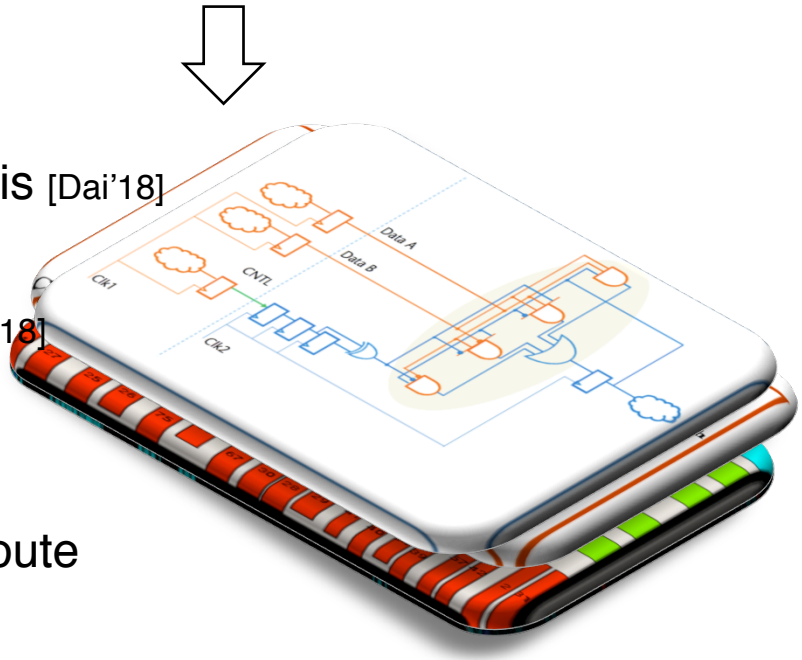
Increasing manual efforts of automations

Let's automate it, again!

Focuses of EDA

- Accomplish IC designs
  - **Design closure**
    - Timing, power, heat, etc.
  - Simulation and verification
  - Signoff

- **Design flows**
  - C/SystemC/HDL
    - e.g., high-level synthesis [Dai'18]
  - Logic/Gate-level netlist
    - e.g., logic synthesis [Yu'18]
  - Transistor-level netlist
  - Physical layout [Yu'19]
    - Floor plan, place and route



Focuses of EDA

- Methodologies
  - Abstraction
    - e.g., Logic synthesis
  - Decomposition

- Advantages
  - Easier R&D process
  - Scalability and Runtime
    - Quality-of-result (QoR)

- Losing correlations to the physical layout

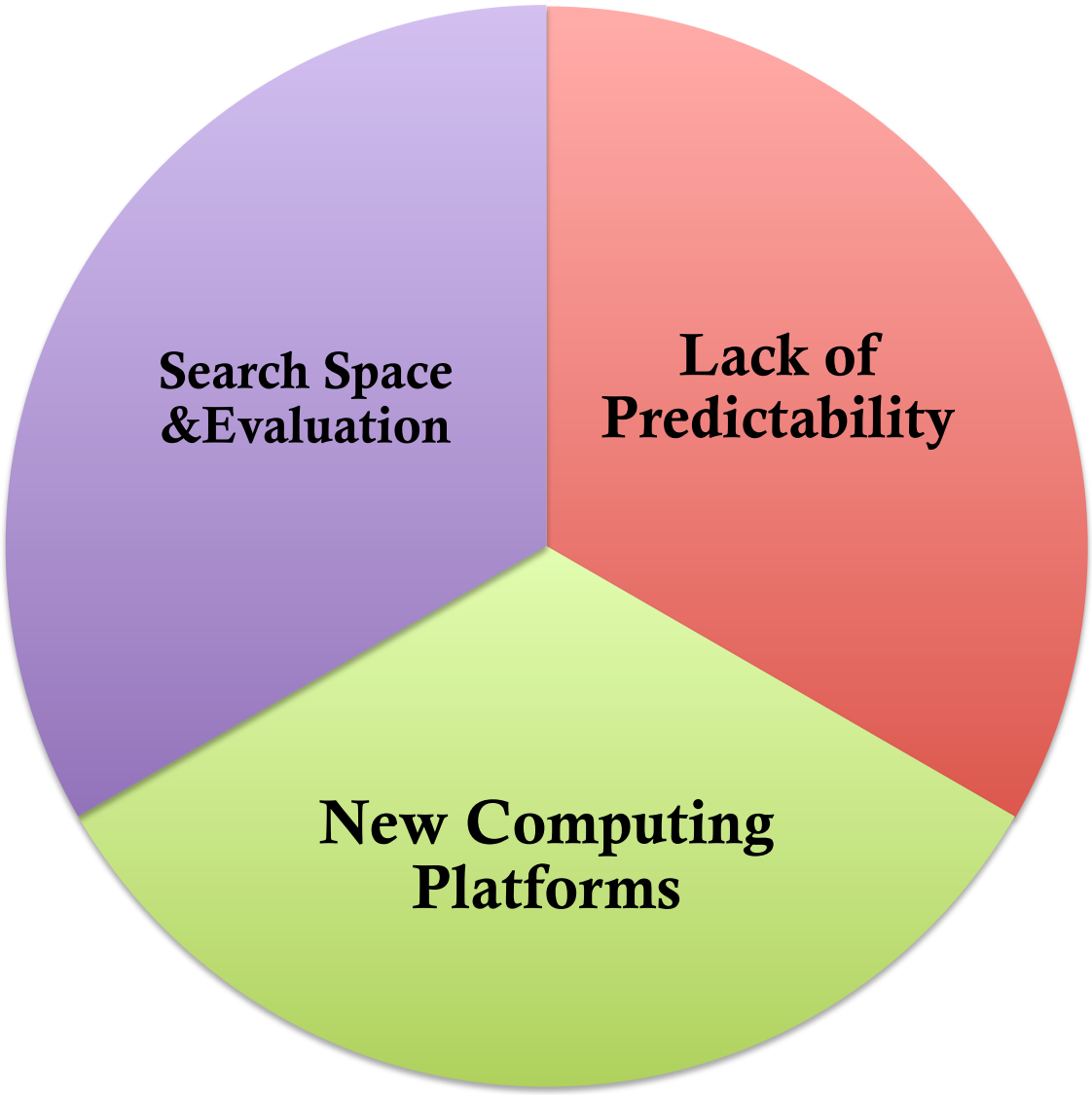
How Difficult Is It?

Step 1: *git clone qflow/vtr*

Step 2: *./configure;make install*

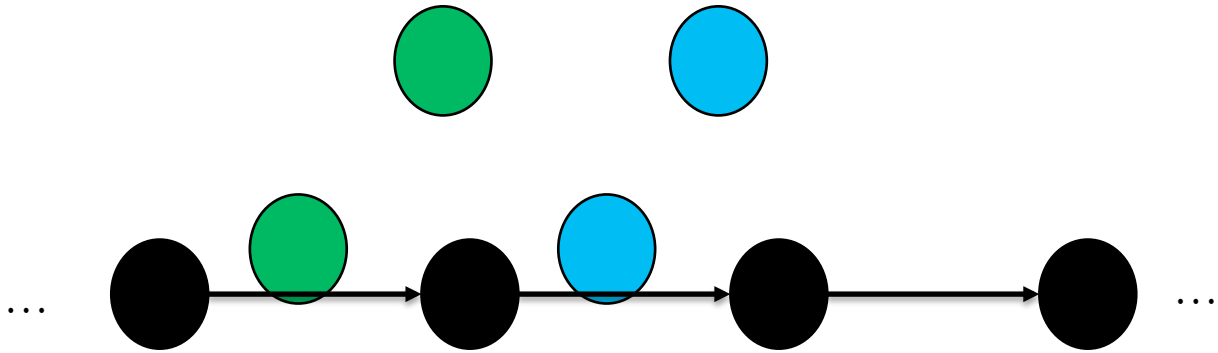
Step 3: run.sh …(load your own design)

Challenges


Search Space
&Evaluation
Lack of
Predictability
New Computing
Platforms

A ‘Failed’ Internship Project

- Internship task: find out *where to use “retiming”?*
  - To reduce the critical path delay
    - Varies more than **150%** (e.g., 4.0 GHz vs. 2.7 GHz)
  - Two retiming options and a fixed order flow
    - A serial design optimization techniques (= 200)
      - 201*202=40,602

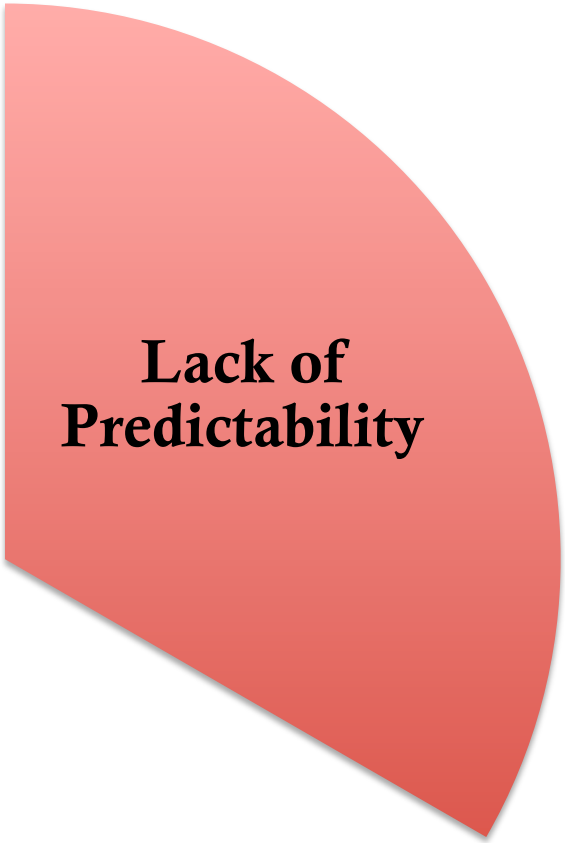


A ‘Failed’ Internship Project

- Internship task: find out *where to use “retiming”?*
  - To reduce the critical path delay
    - Varies more than **150%** (e.g., 4.0 GHz vs. 2.7 GHz)
  - Two retiming options and a fixed order flow
    - A serial design optimization techniques (= 200)
      - 201*202=40,602

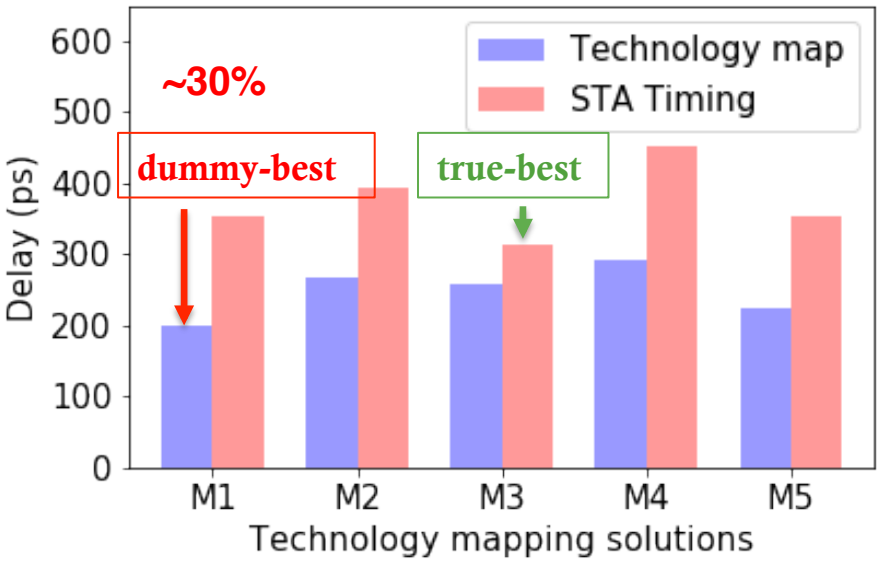
- IBM PowerPC9, split into 7 sub-designs [Yu’18]
  - One evaluation on a single sub-design: **20 hours**
    - Need a week to evaluate all
    - **One year**: 52/40602 = **0.12 %**
    - What if order is flexible ?
  - Algorithmic objective vs. actually timing

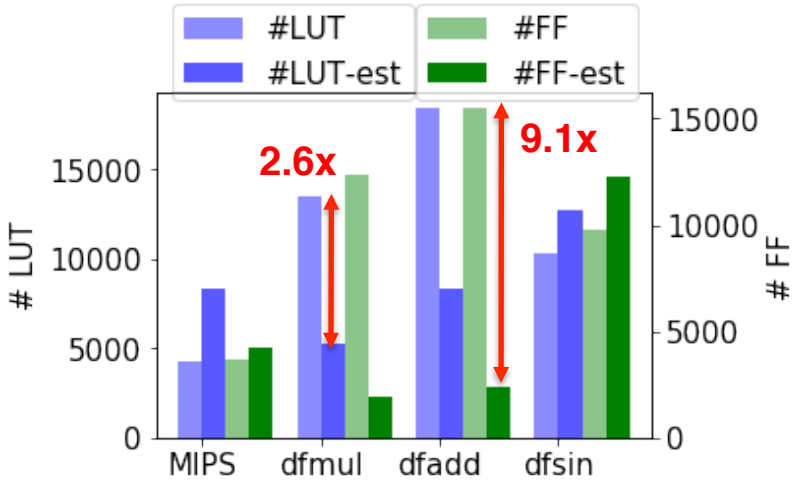
Challenges



Lack of Predictability

- Example of ASIC flow
  - **Cross a single-stage**
    - Gate-level vs. Transistor-level (STA) [Yu'17]
  - Cadence Genus; 14nm



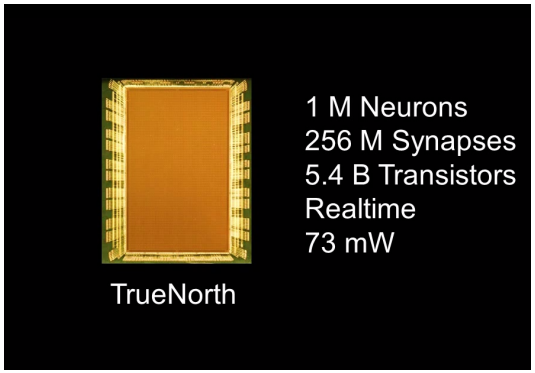


- Example of FPGA flow
  - **Multi-stage**
  - Synthesis estimation vs. post-placement&route
    - #LUT, #FF [Dai'18]
  - Intel Quartus
    - Arria 10 devices

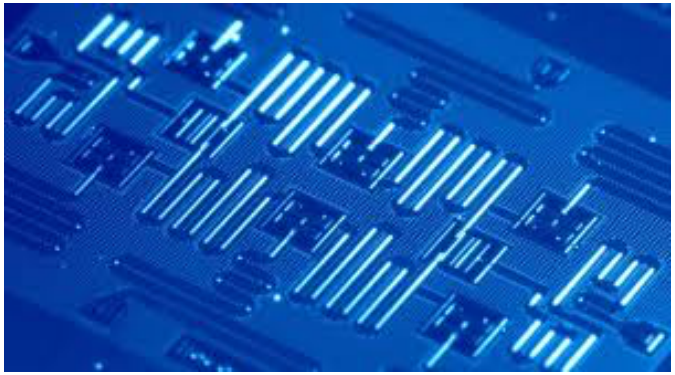
Challenges

Advanced Computing Platforms

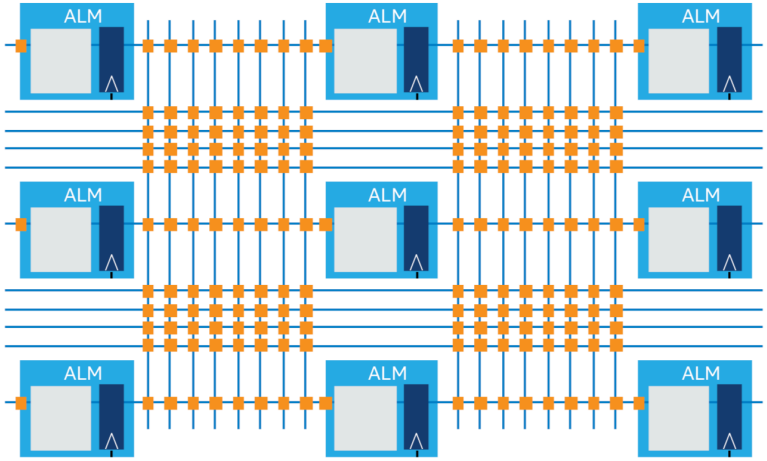
- Challenges in new design styles
  - Neuromorphic chip, Quantum chip, etc.
  - “Slightly” changed architecture
  - **Limited knowledge**



TrueNorth



IBM Q



Intel HyperFlex

Advanced Computing Platforms

- Limited ***time-to-market*** and ***budge***

Mobile

Apple (A12)
Samsung (Exynos 9820)
Huawei (Kirin 970)
Qualcomm (Hexagon DSP)

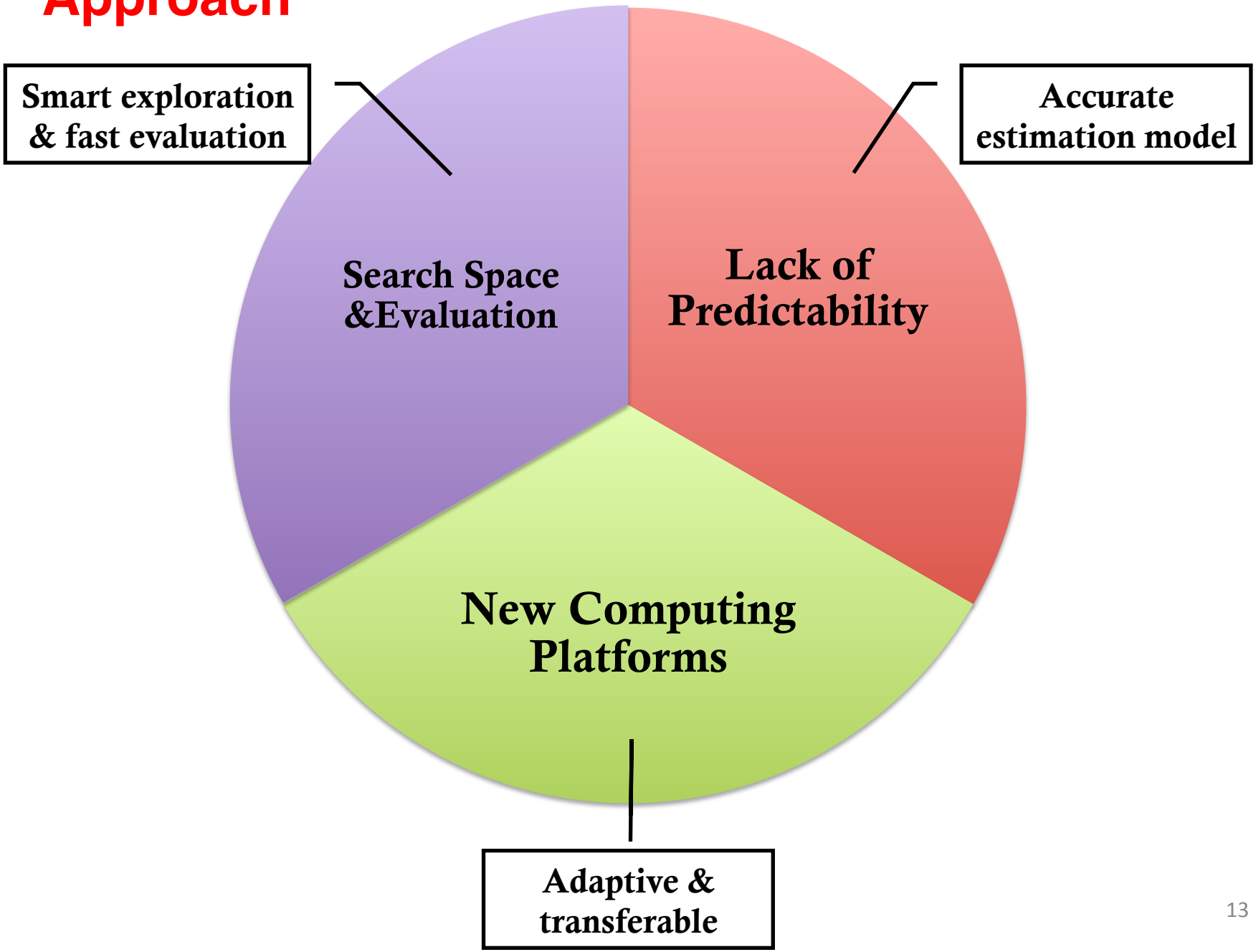
Cloud

NVIDIA (cuDNN)
Google (TPU)
Microsoft (Brainwave)
Xilinx (UltraScale + on EC2 F1)
Intel (Nervana)

Embedded

Google (Edge TPU)
Intel (Movidius)
Deephi/Xilinx (Zynq)
ARM (announced)

Approach

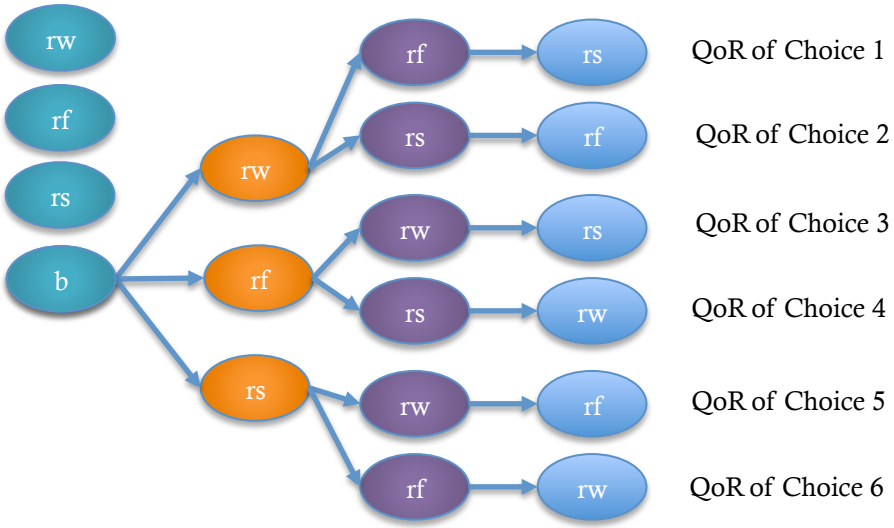

Smart exploration & fast evaluation
Accurate estimation model
Search Space &Evaluation
Lack of Predictability
New Computing Platforms
Adaptive & transferable

Developing Synthesis Flows without Human Knowledge [DAC'18]

Cornell University

Definition and Search Space

- Decision making in design flows
  - Case 1: none-repetition flow
  - Each transformation **appears only once**
  - Example: n=4
    - balance (b), rewrite (rw), refactor (rf), resub (rs)



Search space

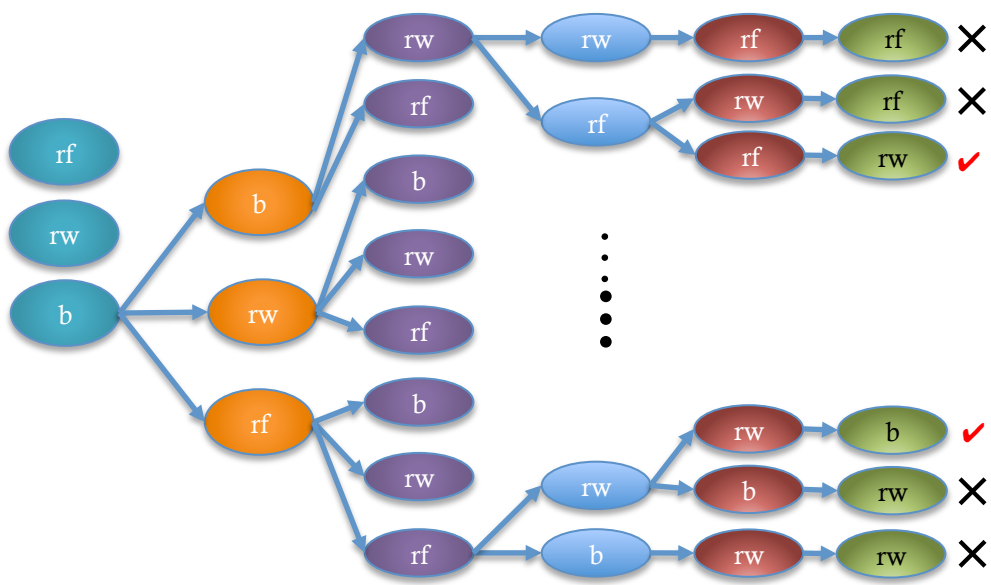
$$f(n) = n!$$

4!=24

Definition and Search Space

- Case 2: *m*-repetition flow
  - Each transformation **appears *m* times**
  - Example: *m*=2, *n*=3
    - balance (b), rewrite (rw), refactor (rf)
    - n=4, m=1, 24

$$\frac{(m_0 + m_1 + \cdots m_{n-1})!}{(m_1!)(m_2!)\cdots(m_{n-1}!)}$$



$$f(n) = \frac{(n \cdot m)!}{(m!)^n}$$

6!/2^3=90

This work

n=6

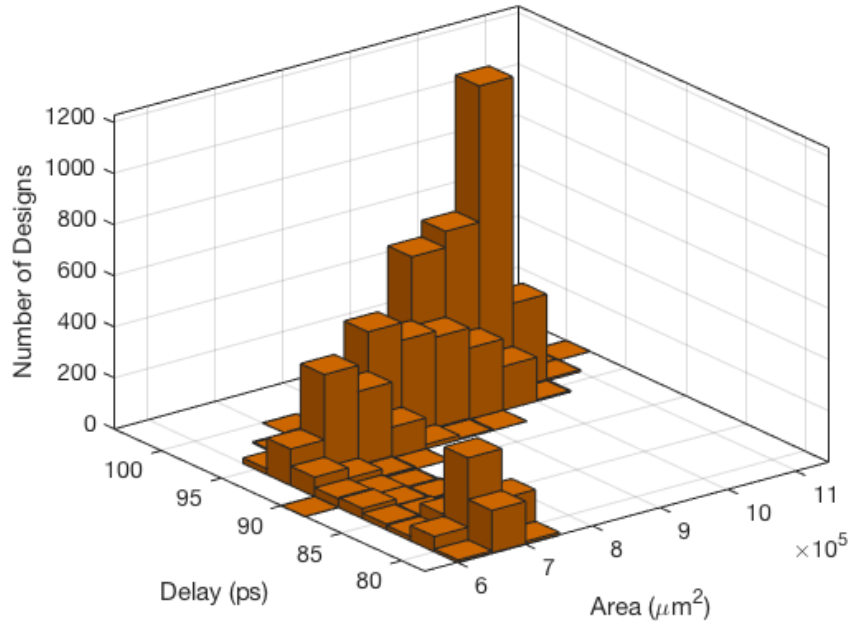
m=4

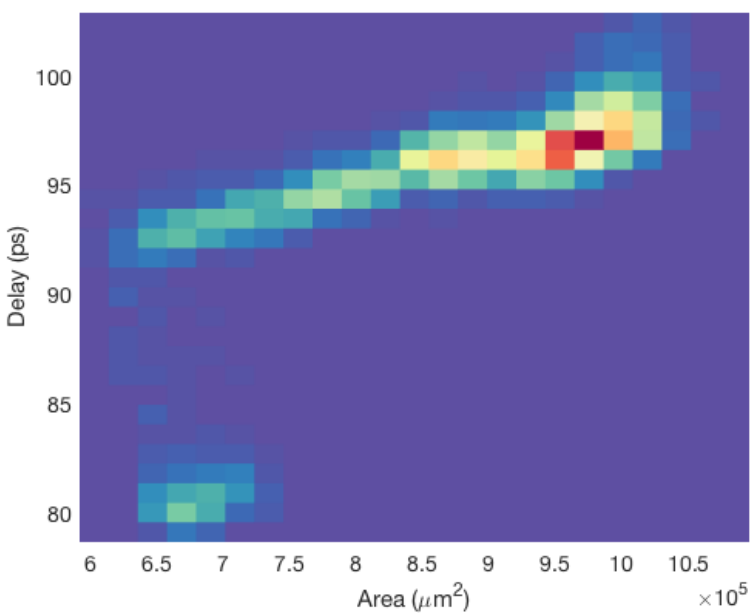
≈

3.25e+15

Example 1 – Permutation of Flows

- 128-bit AES
  - 50,000 random flows
    - Random permutation





  - QoRs collected after tech-map
    - Delay and Area
    - 14nm

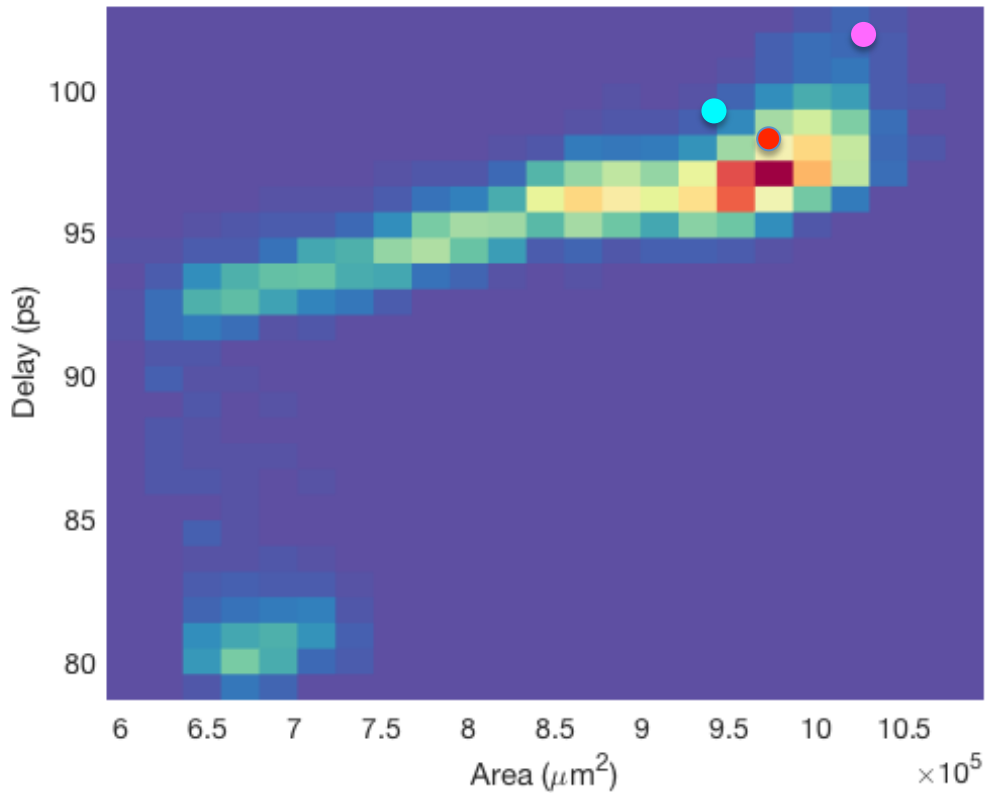
Example 1 – Permutation of Flows

- **Expert-developed** flows

*resyn * 4 (24)*

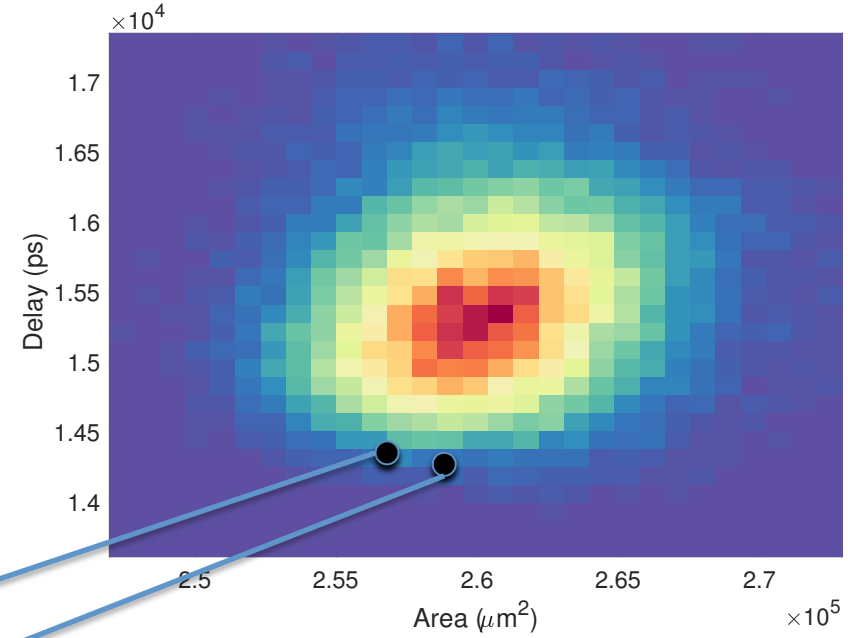
*resyn2 * 4 (40)*

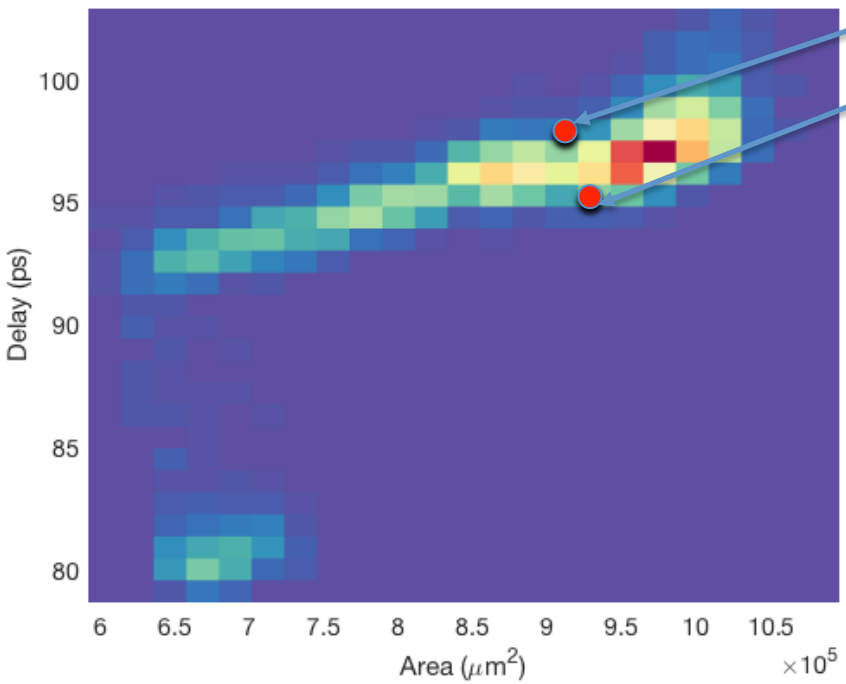
*resyn3 * 4 (36)*



Example 2 – Design Specific

- 128-bit AES vs. 64-bit ALU
  - Same 50,000 flows
  - Statistically significant

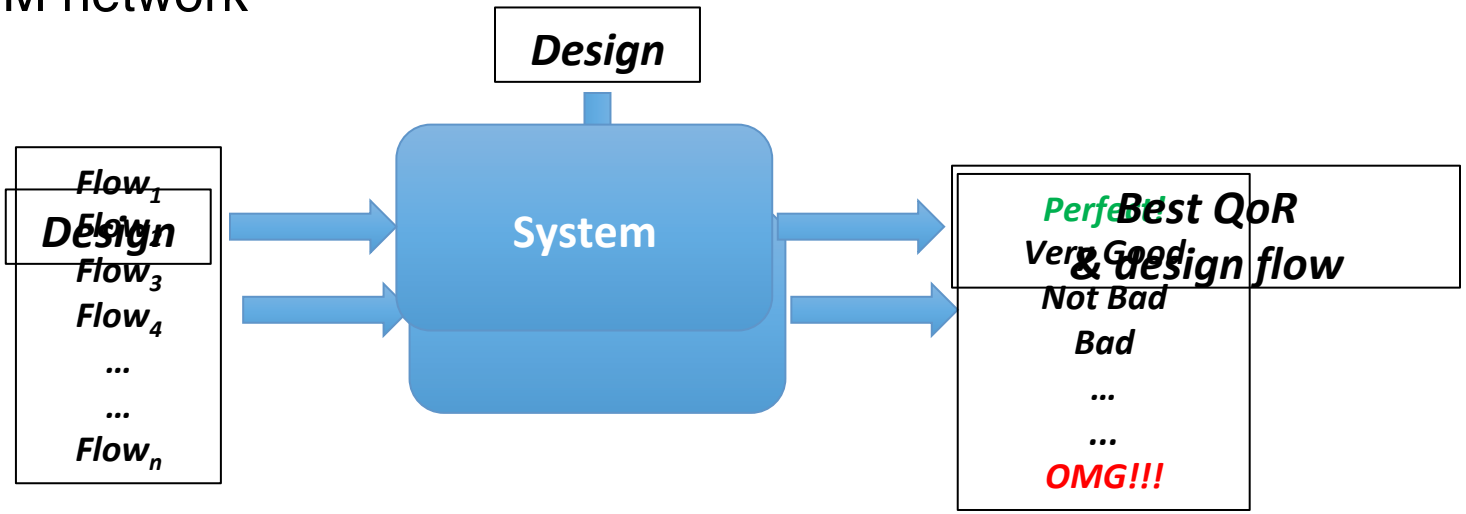




  - Best for ALU ≠ Best for AES

Approach

- Learning-assisted exploration
  - Problem formulation
    - Features and ground truth
  - LSTM network



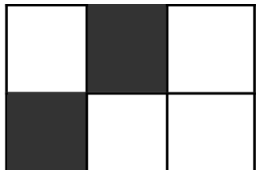
  - Limitation
- Transfer learning
  - Cross technologies and designs

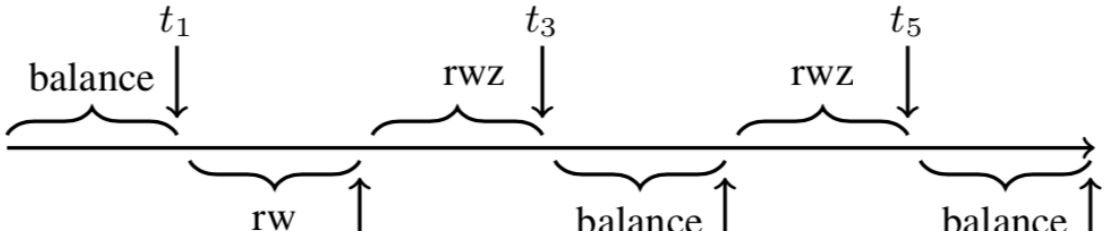
Features - Flow Embedding

- Flow in One-hot Matrix
  - Example: balance (b), rewrite (rw), rw -z (rwz)
    - b = $[1\ 0\ 0]^T$
    - rw = $[0\ 1\ 0]^T$
    - rwz= $[0\ 0\ 1]^T$
      - Flow: rw -> b -> rwz
  - ***e.g., default flow: resyn***
    - ***When n=6, m=4***

rw -> b -> rwz

$$\begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \end{bmatrix}$$

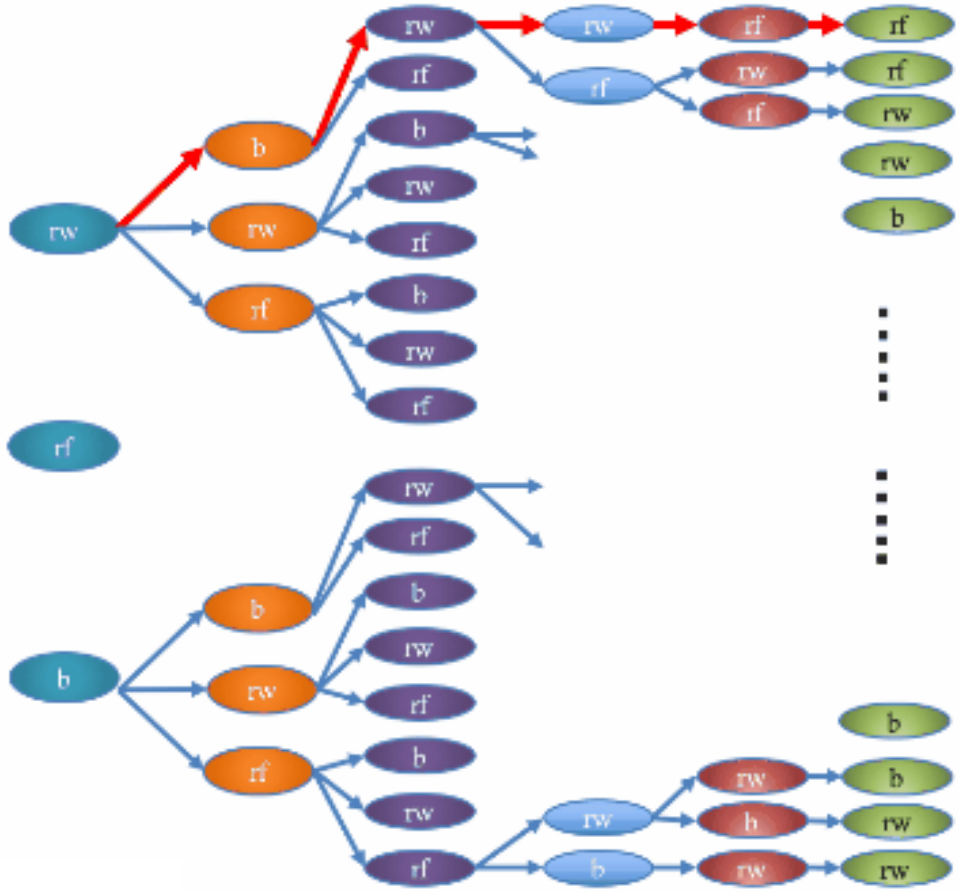


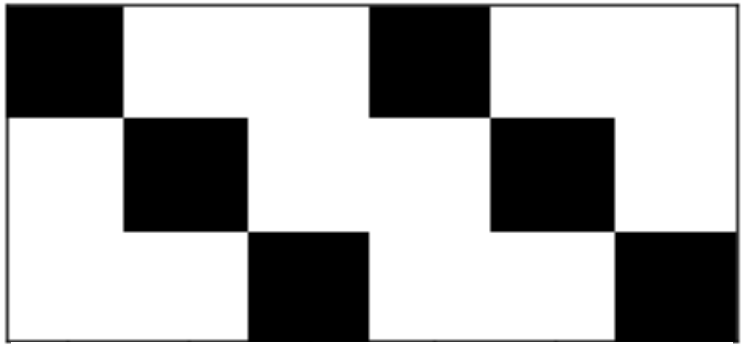


t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}	t_{11}	t_{12}	t_{13}	t_{14}	t_{15}	t_{16}	t_{17}	t_{18}	t_{19}	t_{20}	t_{21}	t_{22}	t_{23}	t_{24}
b	rf	rwz	rw	rs	rfz	b	rw	rwz	rf	rs	rfz	b	rw	rwz	rw	rs	rfz	b	rf	rwz	rfz	rs	rf
			■				■						■		■								
		■						■						■						■			
■						■						■						■					
				■						■						■						■	
					■						■						■				■		
	■								■										■				■

Features - Flow Embedding

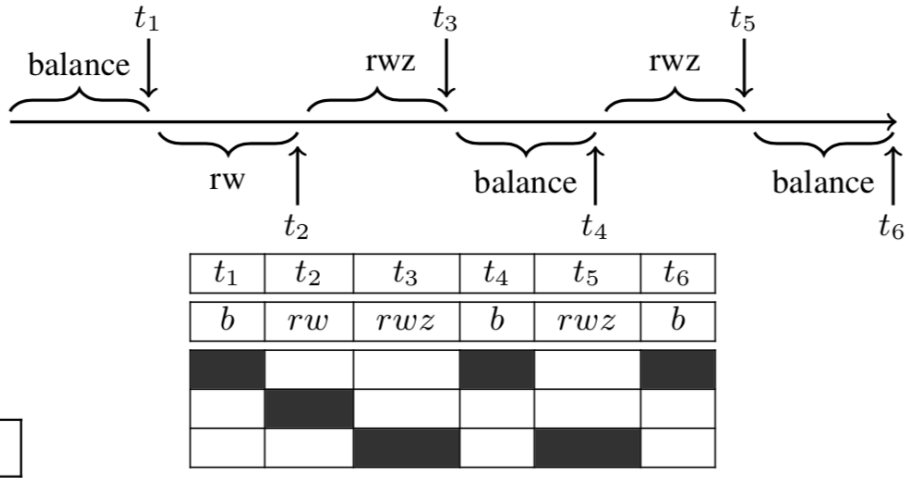
- Decisions to Matrices
  - One-to-one mapping: decision path -> matrix





LSTM Network and Transfer Learning

- LSTM network
  - Features
  - Labels
    - **Delay, area**, power, etc.

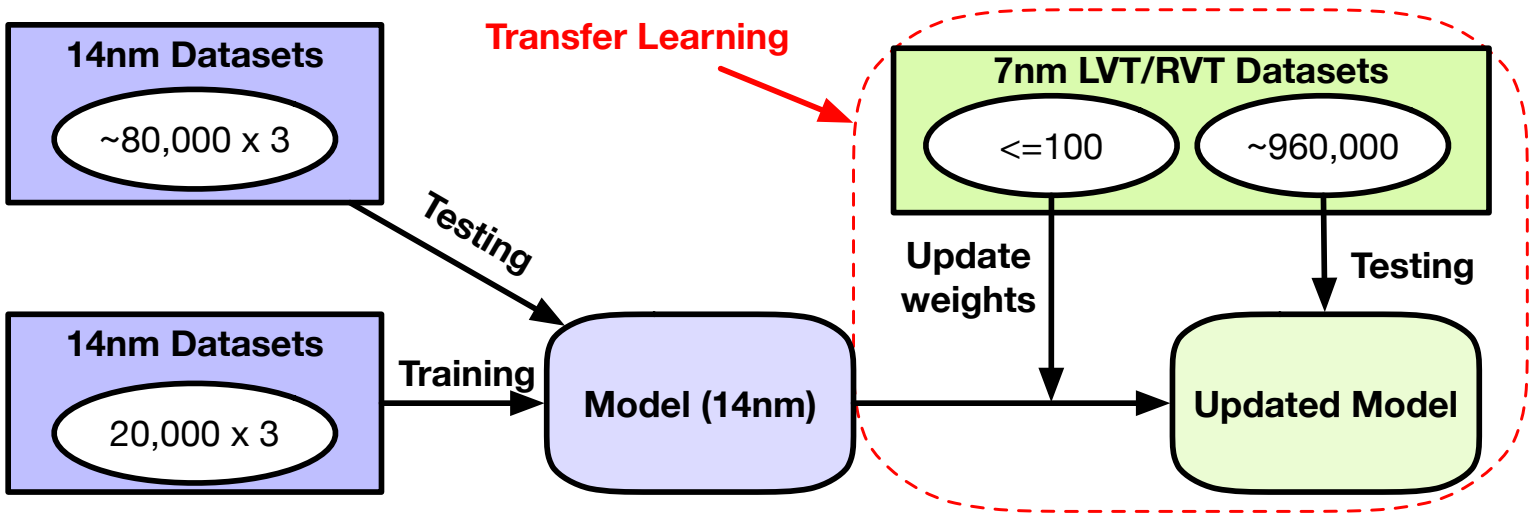


Layer : Type	Output Shape	# Param
1:LSTM1	(None, 24, 128)	68608
2:BN1	(None, 24, 128)	512
3:LSTM2	(None, 128)	131584
4:BN2	(None, 128)	512
5:Dense1	(None, 30)	3870
6:BN3	(None, 30)	120
7:Dense2	(None, 30)	930
8:BN4	(None, 30)	120
9:Dropout	(None, 30)	0
10:Dense3	(None, dim)	$31 \times dim$

- Transfer learning
  - Data collection is extremely time-consuming
  - **New designs** and **technology nodes**

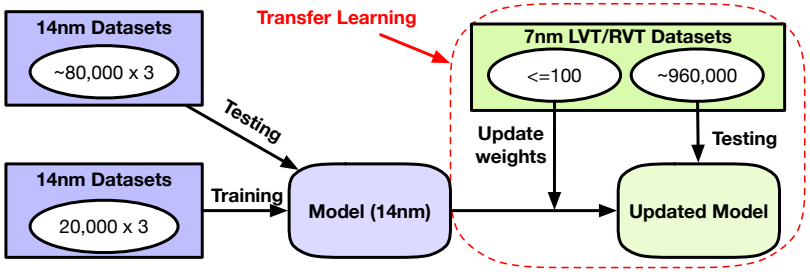
LSTM Network and Transfer Learning

- Transfer-learning
  - Initial model trained on 14nm
  - Update model with **<=100** data
    - **1) Update entire model**
    - 2) Update 2xDense only
    - ~~3) Update 2xLSTM only~~



Dataset

- Extended dataset
  - Global Foundries (IBM) 14nm FinFET
  - ARM 7nm FinFET
    - *RVT* = Regular voltage transistor
    - *LVT* = Low voltage transistor
  - 100k or 20k applied to each design

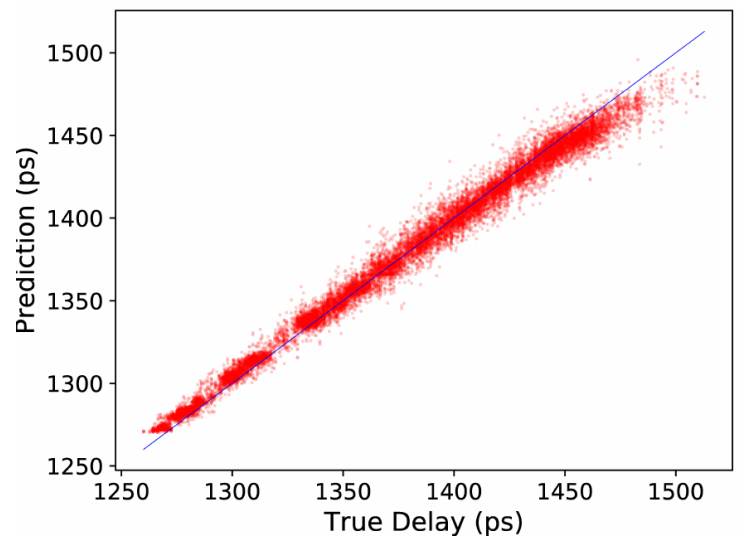


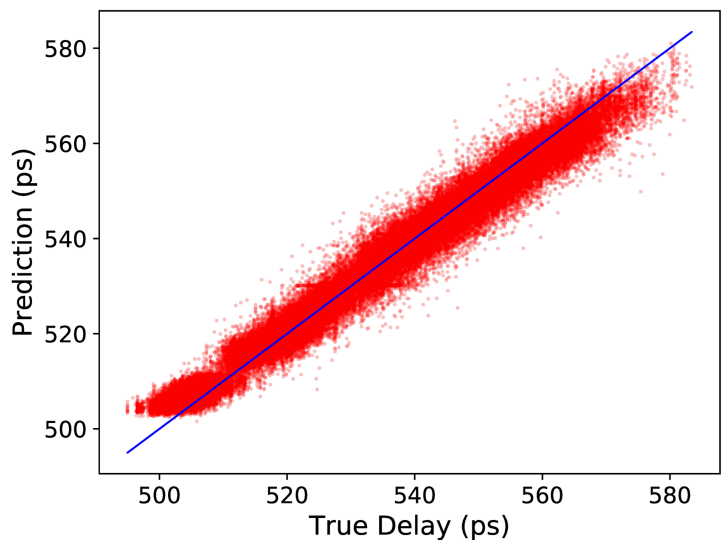
Design	14nm	7nm RVT	7nm LVT
64-bit Montgomery	99,997	99,997	99,997
64-bit ALU	100,000	100,000	100,000
128-bit AES core	99,737	99,737	99,737
LU8PEng	-	20,000	20,000
Stereovison0	-	20,000	20,000
Stereovison1	-	20,000	20,000
SHA	-	20,000	20,000
raygentop	-	20,000	20,000
OR1200	-	20,000	20,000
Boundtop	-	20,000	20,000
blob_merge	-	20,000	20,000
bgm	-	20,000	20,000

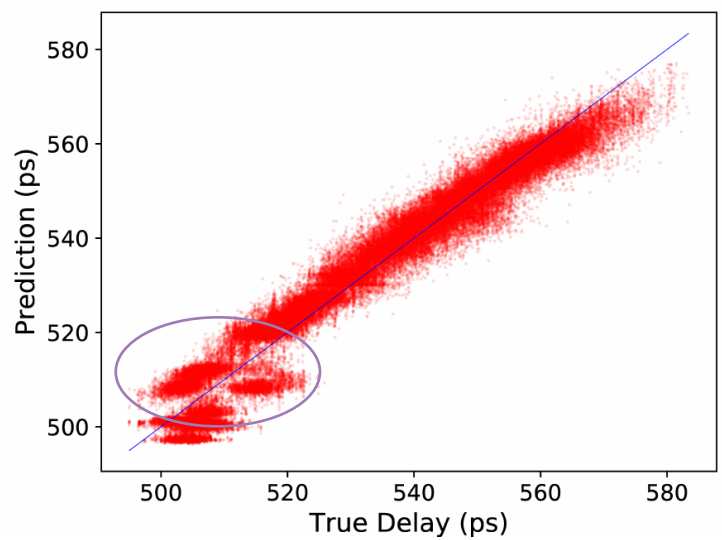
Another example of "lack of predictability"!!

Evaluation - Transfer Learning

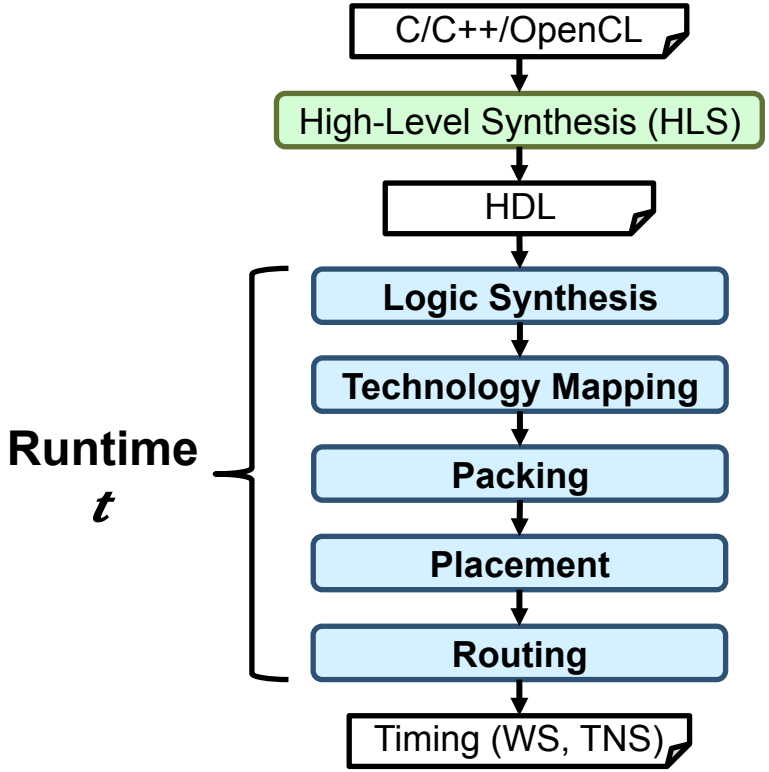
- Initial model (14nm)
  - Training with 20k **14nm** dataset
  - Test with 80k **14nm** dataset
- Transfer learning
  - **100** new **7nm** data points
  - Test with **7nm** dataset
    - Cross technologies
    - Cross designs



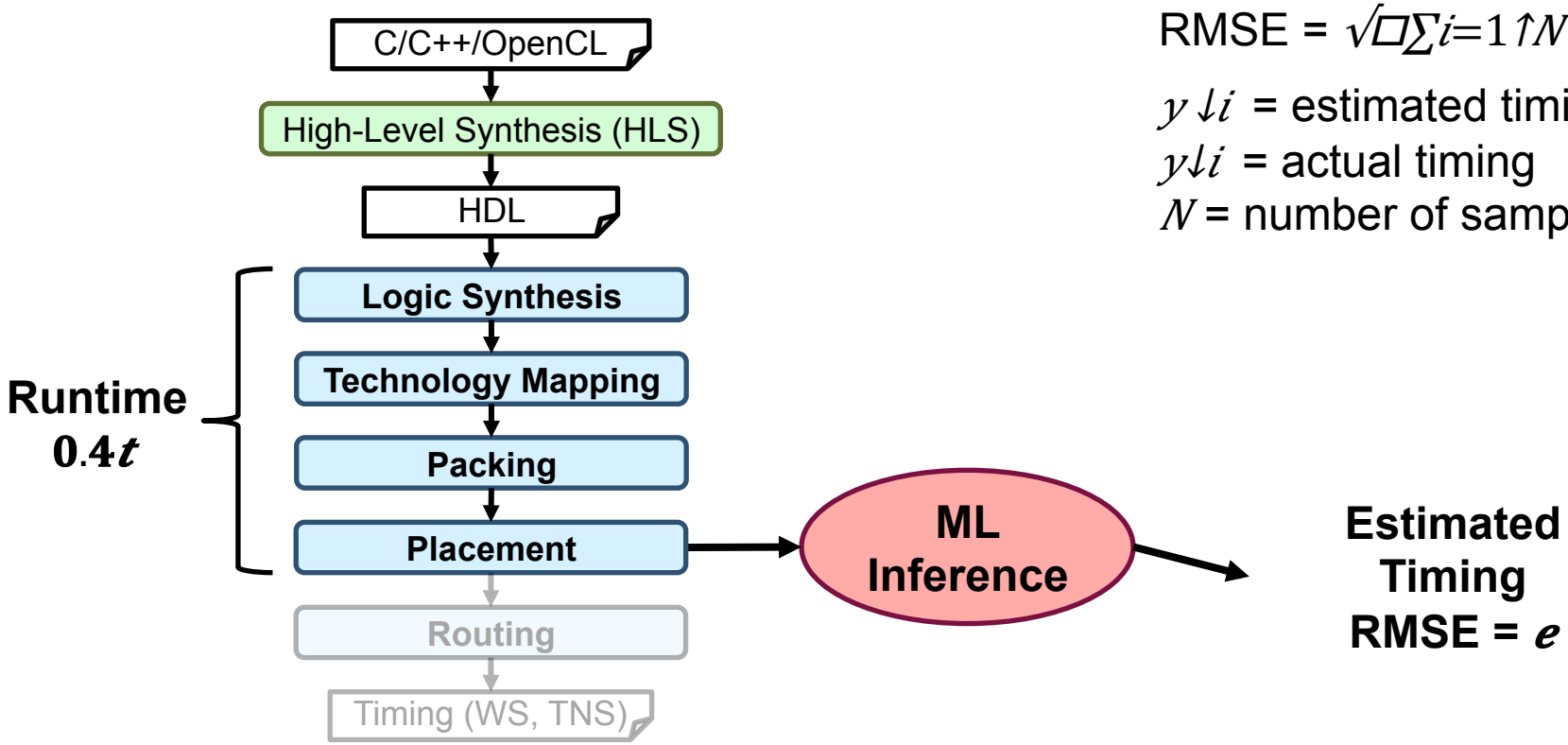




Leveraging Multi-Stage ML Inference



Leveraging Multi-Stage ML Inference



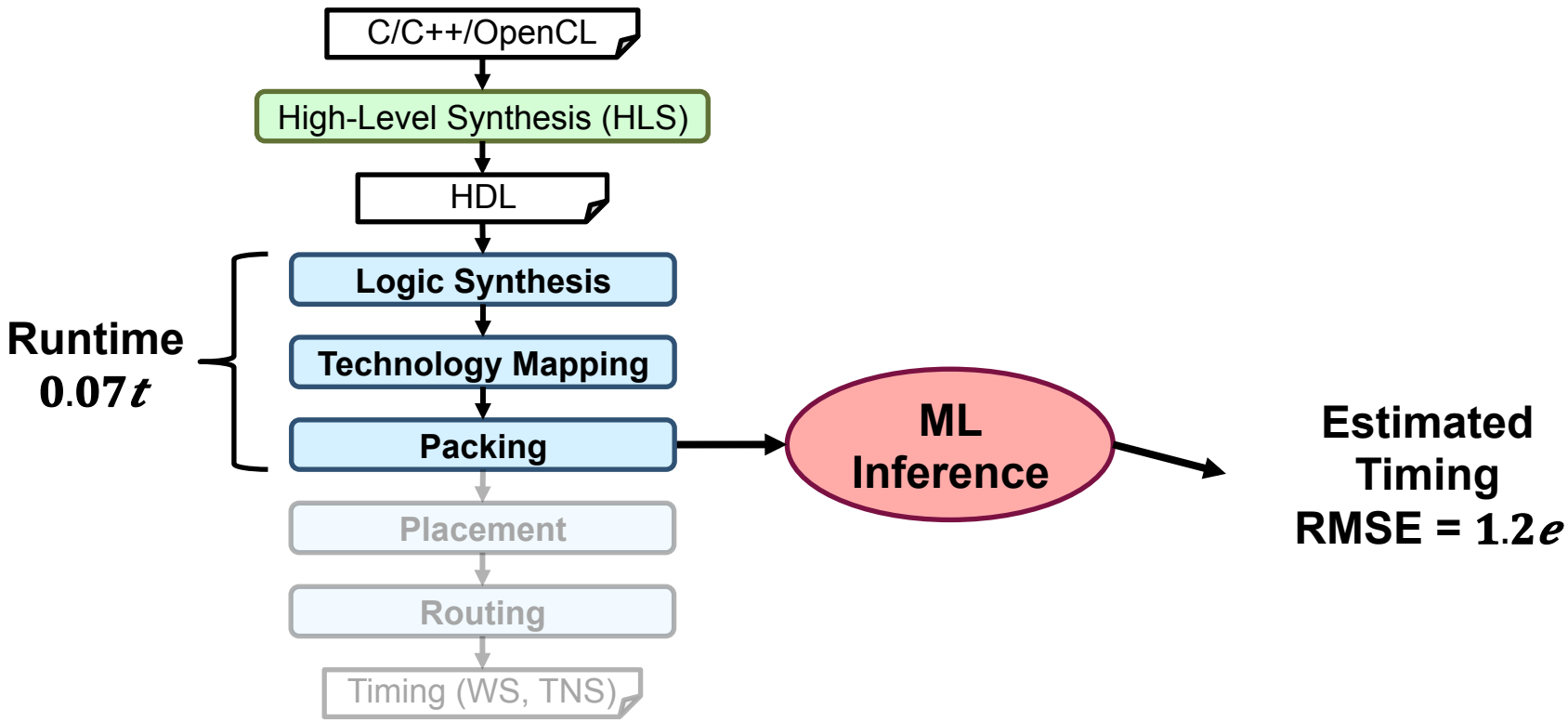
RMSE = $\sqrt{\frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2}$

y_i = estimated timing

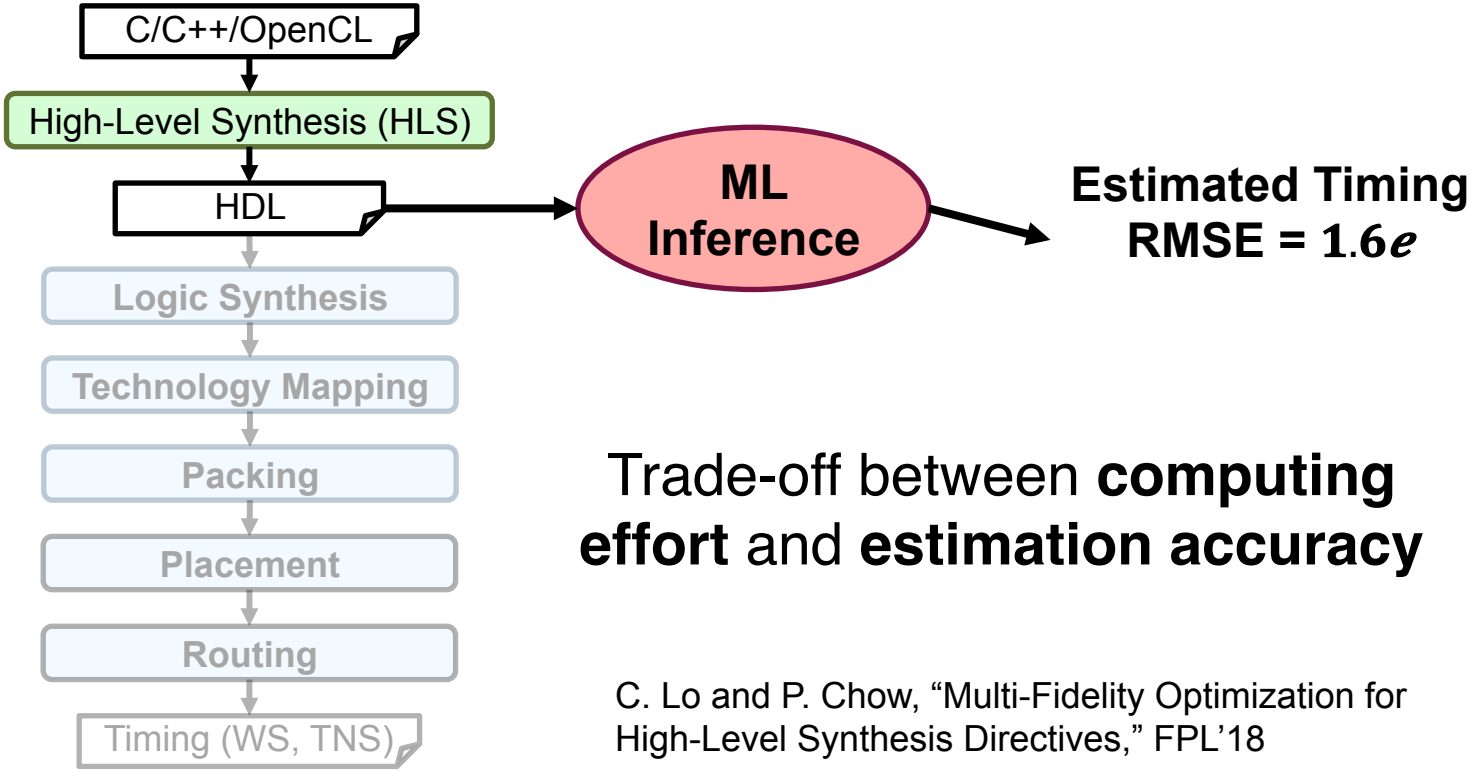
$\hat{y}_i$ = actual timing

N = number of samples

Leveraging Multi-Stage ML Inference



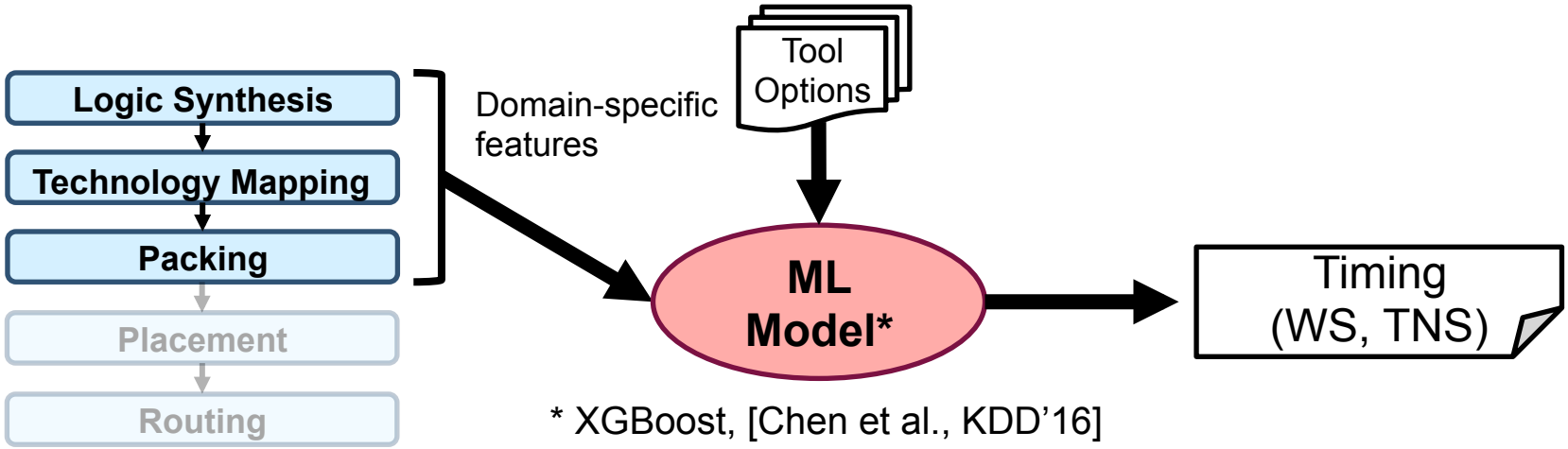
Leveraging Multi-Stage ML Inference



Trade-off between **computing effort** and **estimation accuracy**

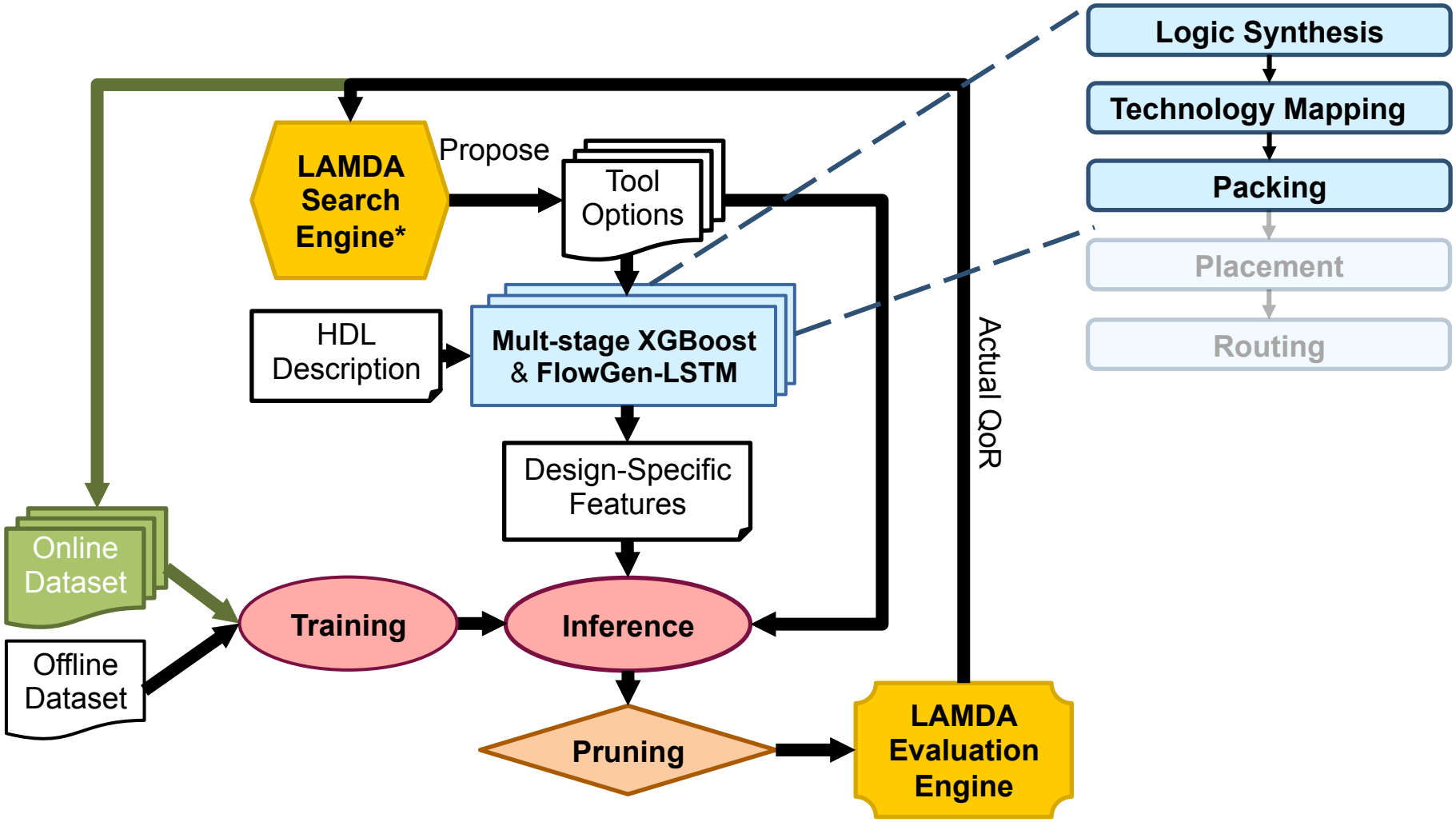
C. Lo and P. Chow, "Multi-Fidelity Optimization for High-Level Synthesis Directives," FPL'18

Leveraging Multi-Stage ML Inference

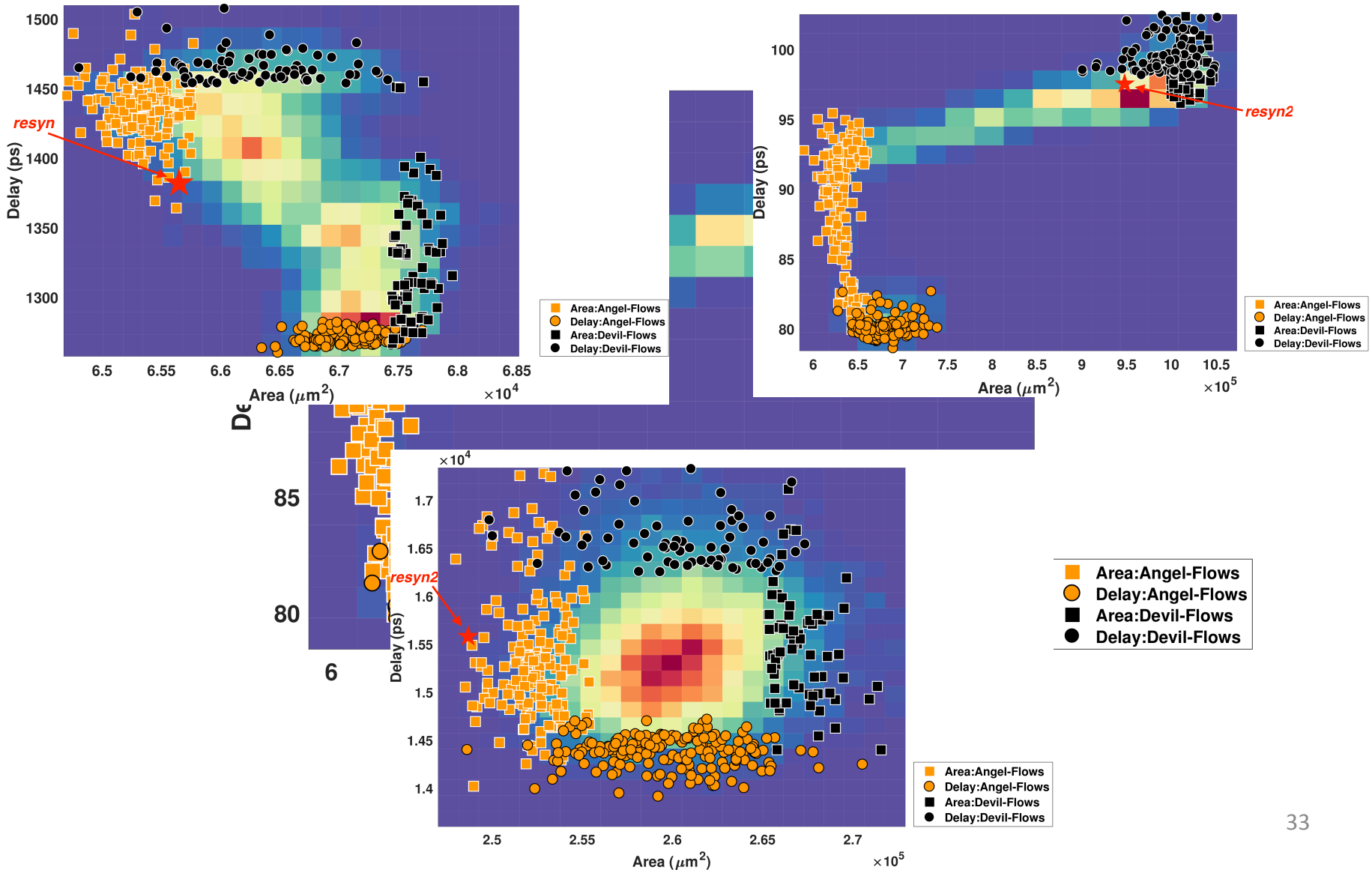


Design Stage	Feature Type	Design-Specific Features
Logic Synthesis, Technology Mapping, and Packing	Resource	#ALM, #LUT, #registers, #DSP, #I/O pins, #fan-out, etc.
	Timing	WS, TNS

LAMDA: Learning-Assisted Multi-Stage Design Autotuning



Results


resyn
Delay (ps)
Area (μm^2)
$\times 10^4$
Area:Angel-Flows
Delay:Angel-Flows
Area:Devil-Flows
Delay:Devil-Flows
resyn2
Delay (ps)
Area (μm^2)
$\times 10^5$
Area:Angel-Flows
Delay:Angel-Flows
Area:Devil-Flows
Delay:Devil-Flows
resyn2
Delay (ps)
Area (μm^2)
$\times 10^4$
$\times 10^5$
Area:Angel-Flows
Delay:Angel-Flows
Area:Devil-Flows
Delay:Devil-Flows
Area:Angel-Flows
Delay:Angel-Flows
Area:Devil-Flows
Delay:Devil-Flows

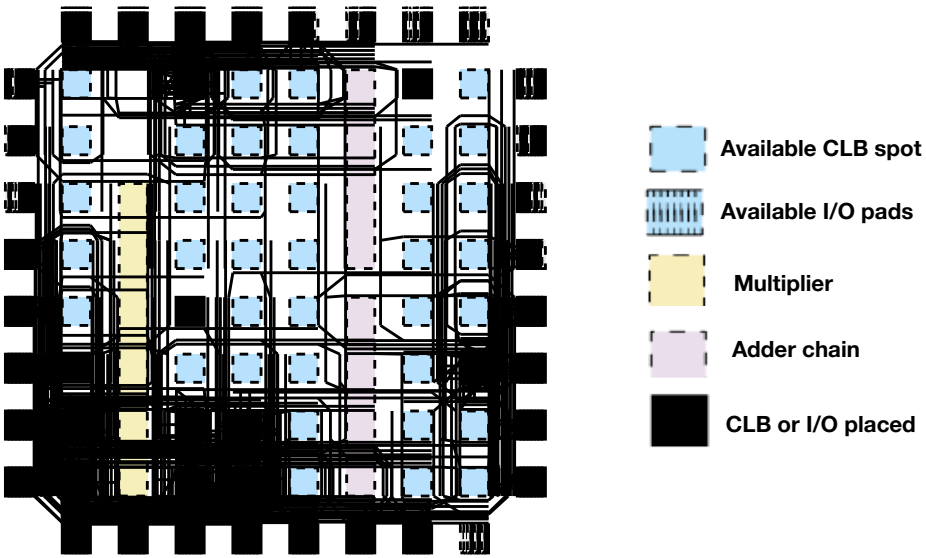
Painting on Placement

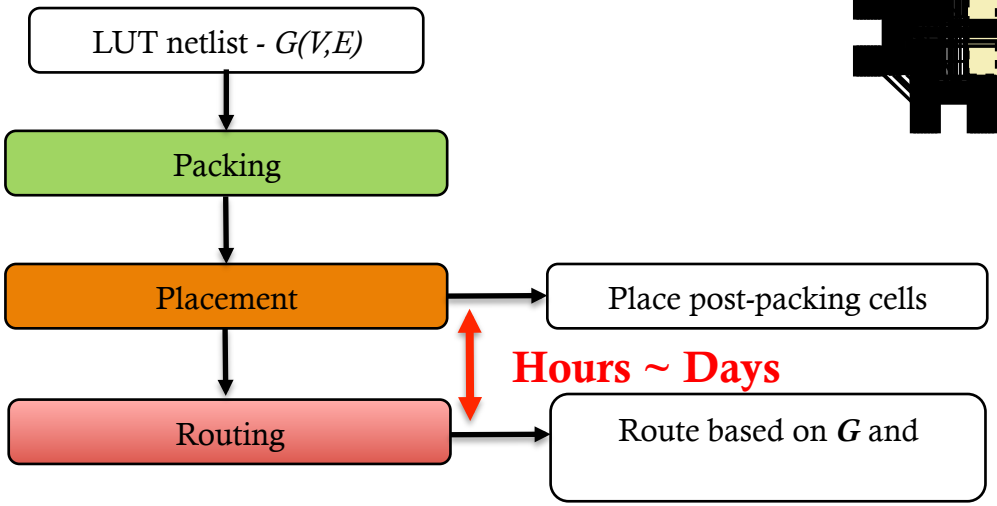
Forecasting Routing Congestion using Conditional GANs

[DAC'19]

Introduction

- Physical design
  - Logic netlist to layout
    - Input: directed graph
    - Output: physical netlist

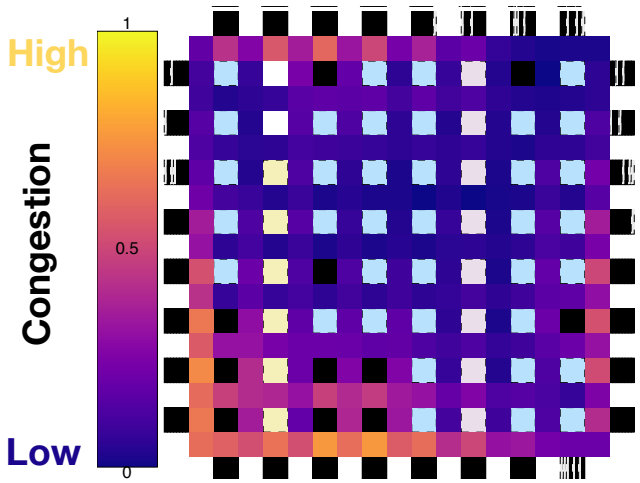




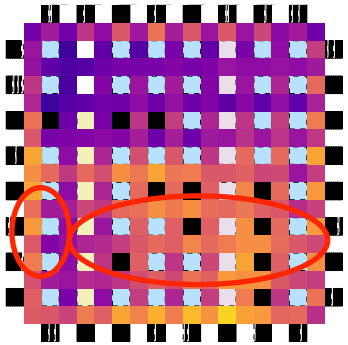
- Routing congestion
  - Utilization(density) of routing channels
- Challenges to predict congestion
  - Routing is extremely time-consuming
  - Challenging to predict congestion

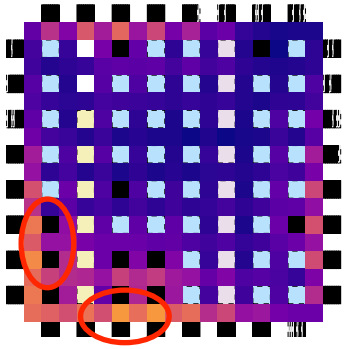
Routing Congestion

- Why congestion is important?
  - Routability
  - Timing closure
- How is congestion evaluated?
  - Visualization of routing congestion
- **Placement plays big role**
  - e.g., 255 MHz vs. 367 MHz



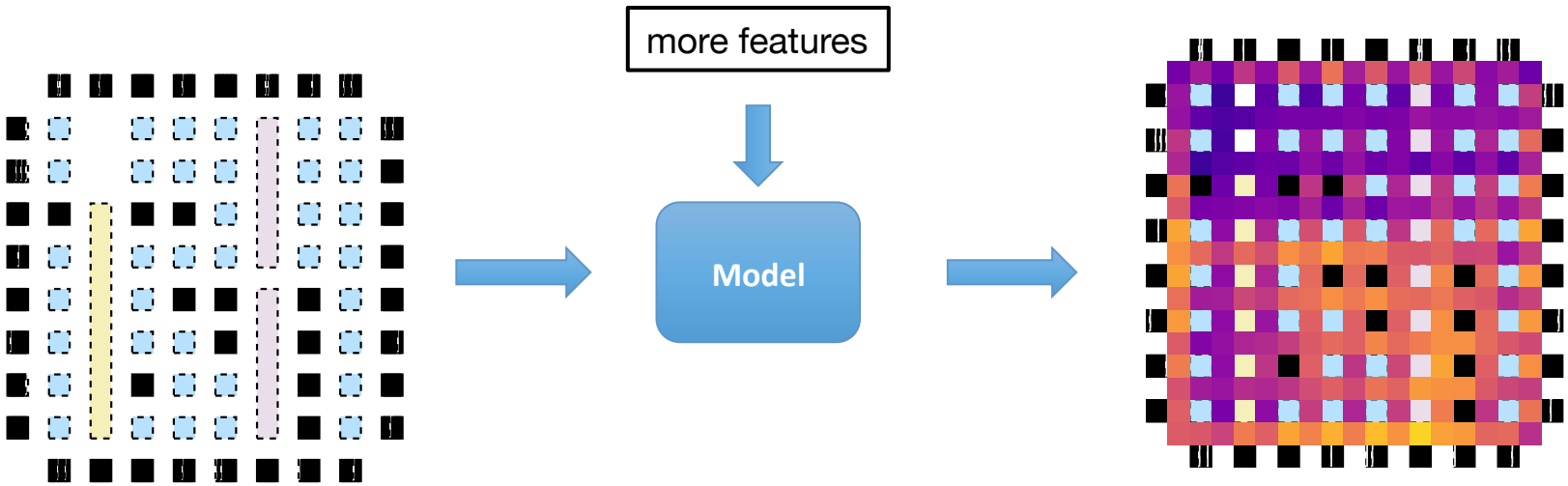
Very time-consuming to get post-routing congestion.





Approach – Painting on Placement

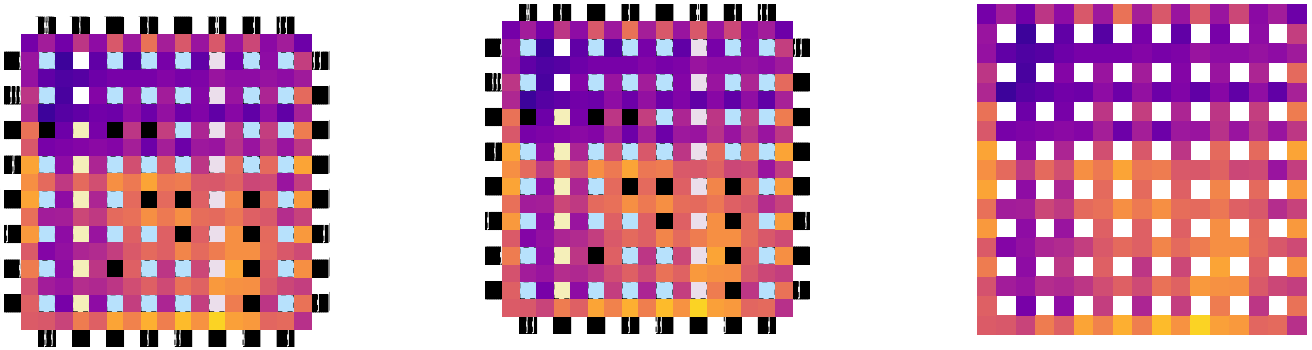
- Forecast congestion from placement
  - Learning a image-to-image mapping [Isola et al, CVPR'17]
    - Placement, congestion heatmap and other features can be represented as images



Approach – Painting on Placement

- Forecast congestion from placement
  - Learning a image-to-image mapping [Isola et al, CVPR'17]
    - Placement, congestion heatmap and other features can be represented as images

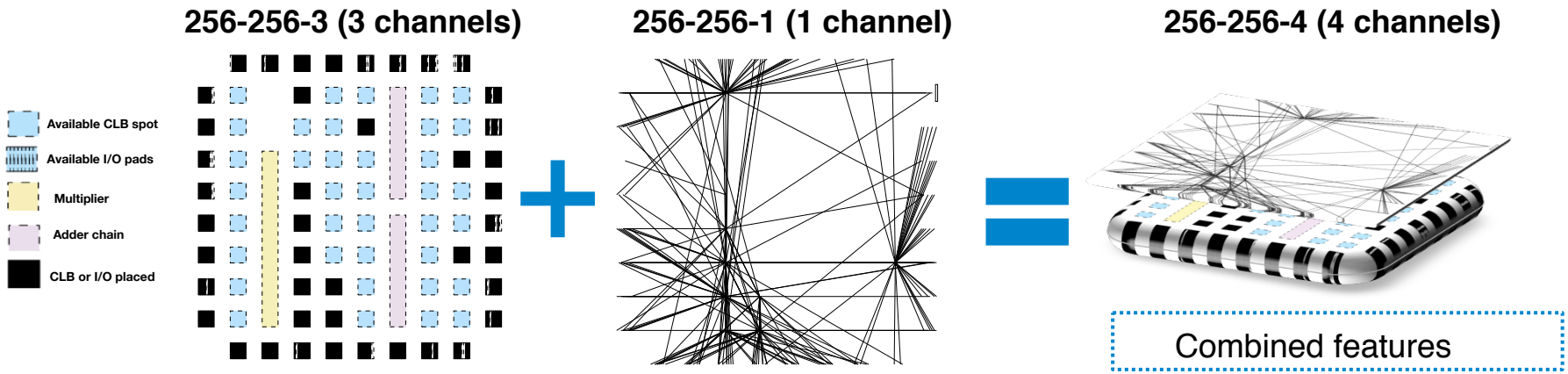
- Forecast as *image colorization*
  - **Colorizing the routing channels of the placement image**
    - Exact the same underlying structure



img_{place} img_{route} $img_{route}-img_{place}$

Features

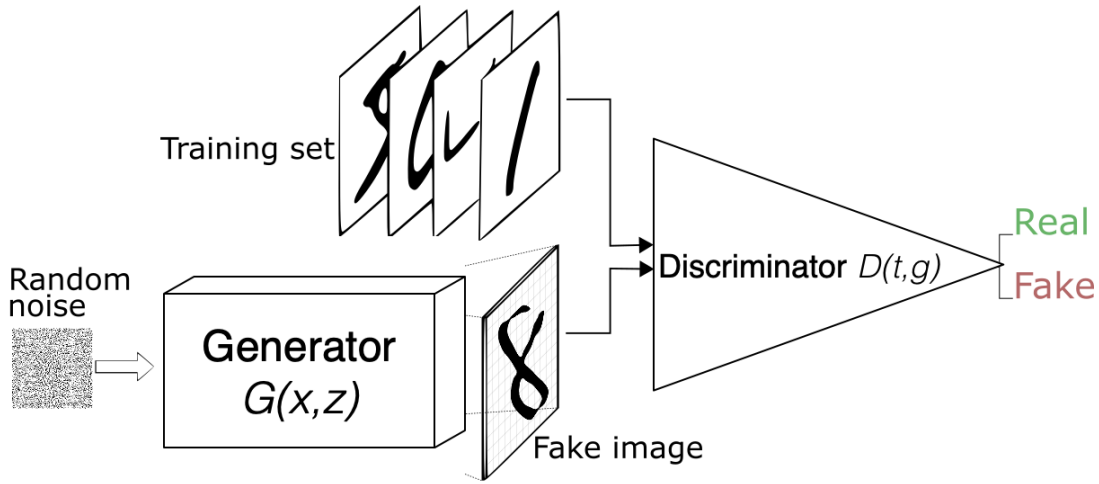
- Embedding of placement and G(V,E)
  - **(1)** Placement image (RGB)
  - **(2)** Visualized graph after placement (grayscale)
  - **Final input**: stack as a four-channel matrix (e.g., 256-256-4)



- Important notes for constructing feature images
  - **Color code** to distinguish different types of cells
  - Adjust **resolution** to distinguish each cell as a objective

Generative Adversarial Nets

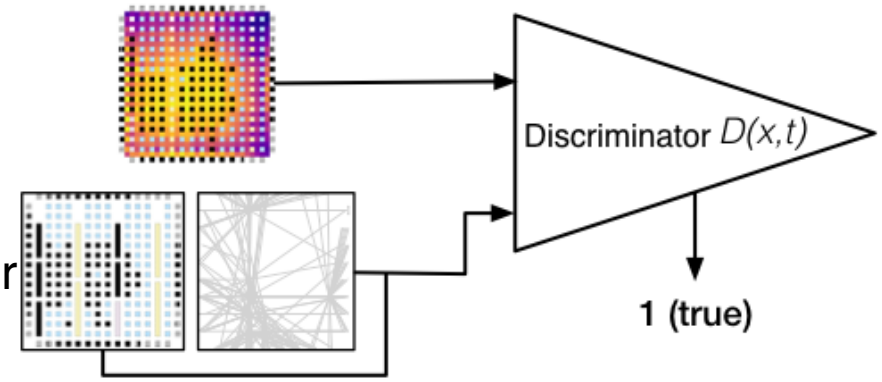
- Generative Adversarial Nets (GAN) [Goodfellow NeurIPS14]
  - Discriminator: learns to **classify** true or fake
  - Generator: learns to **fool** discriminator
  - No control to the generated data

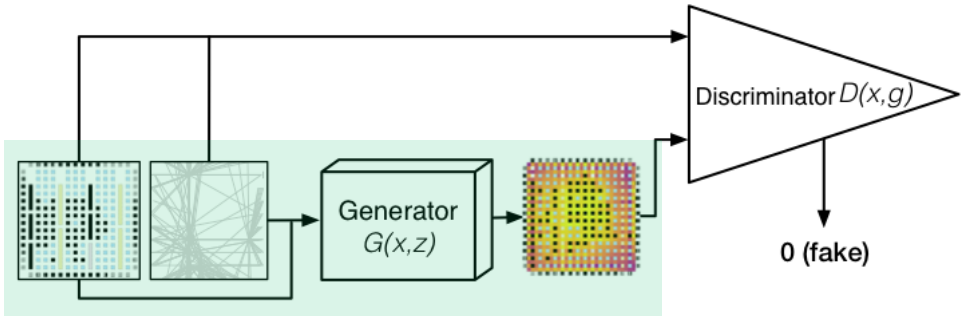


- **Conditional GAN (cGAN)** [Mirza arxiv14]
  - Adding the additional parameter to control the generator
  - Model used in this work
    - G: Fully Convolutional Networks (FCN) with 7 down/up-sampling layers
    - D: CNN based binary classifier

Training and Inference

- Training
  - Discriminator
    - Learns to **classify** true or fake
  - Generator
    - Learns to **fool** the discriminator
  - Input-output pair
    - Stacked matrix
      - 256-256-4
    - Congestion heatmap
      - 256-256-3





- Inference
  - Run generator

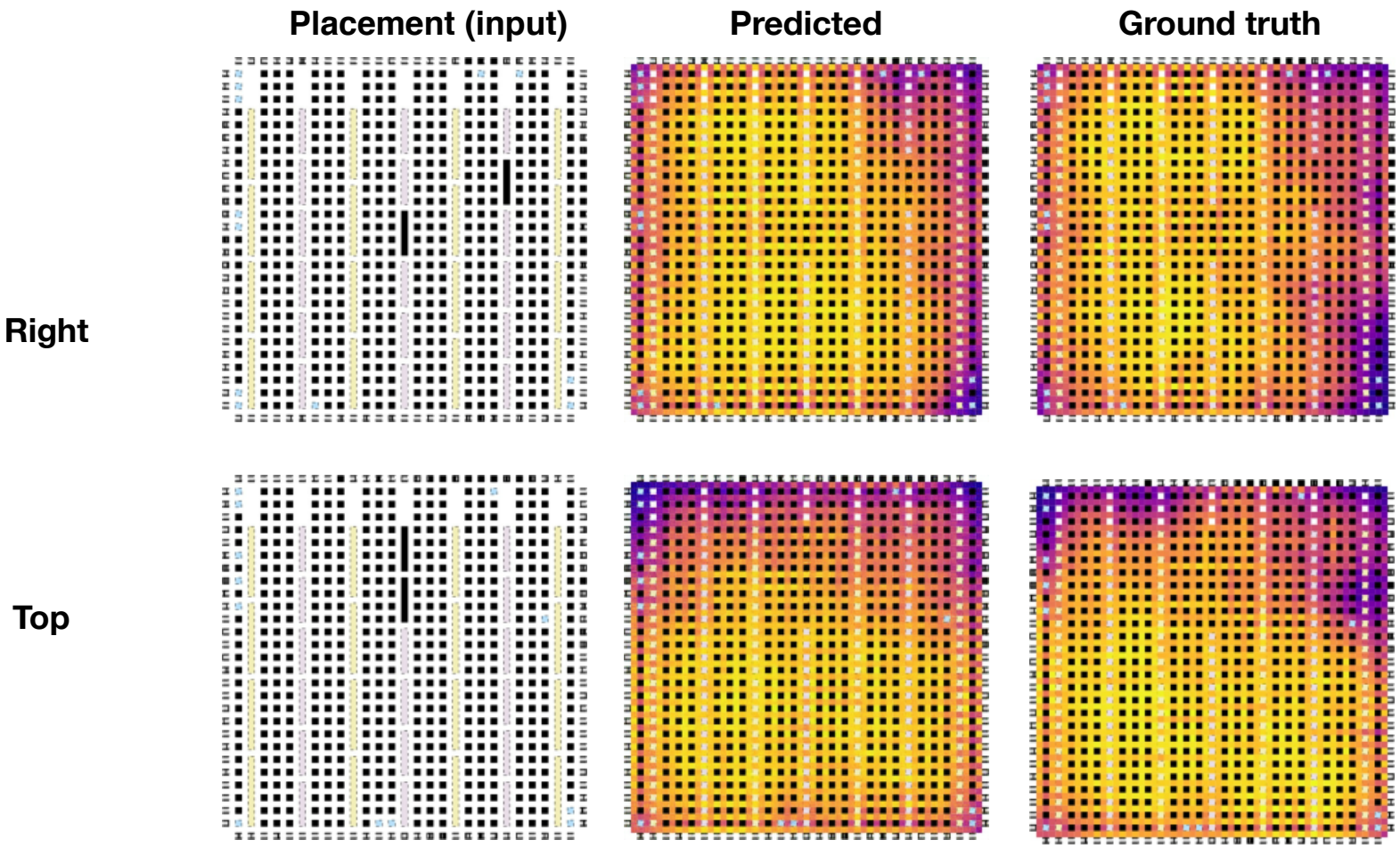
Dataset and Results

- Image generator implemented based on VPR
  - Training: 4x1080Ti, < 3 hours (batch=1)
  - Inference: < 0.2 s on 1x1080Ti
- Min-congestion exploration

Design	# LUTs	# FFs	# Nets	# samples	Top10
diffeq1	563	193	2059	200	**50%**
diffeq2	419	96	1560	200	**40%**
SHA	2501	1047	5023	200	**70%**
OpenRISC	2823	911	10910	200	**40%**
ODE	5488	670	12336	150	**70%**
bfly	9503	1748	38582	150	**80%**
dcsg	9088	1618	36912	150	**70%**

Results – Placement Exploration

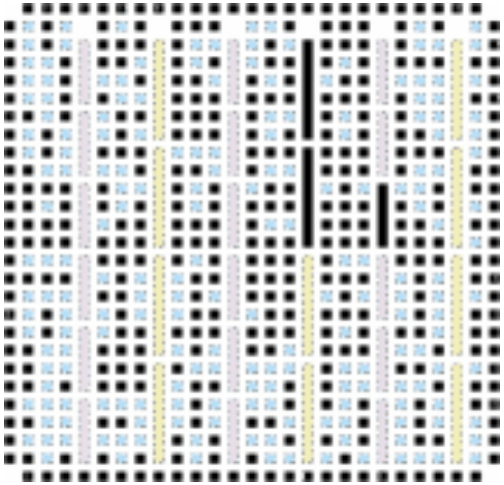
- Constrained placement exploration
  - Placement with biased routing congestion region



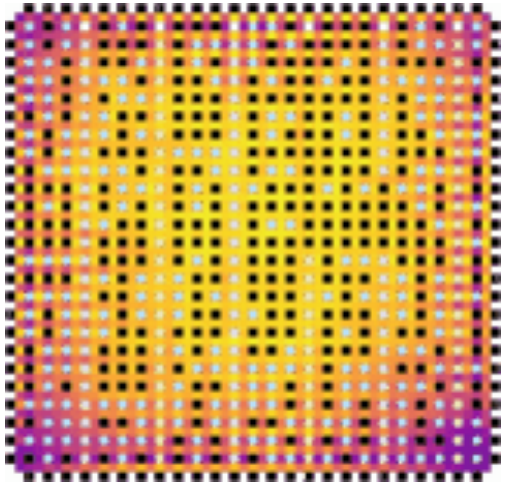
Results – Real-time Forecast

- Visualization of *simulated annealing* placement
  - Swap locations iteratively
  - Every 200 iterations

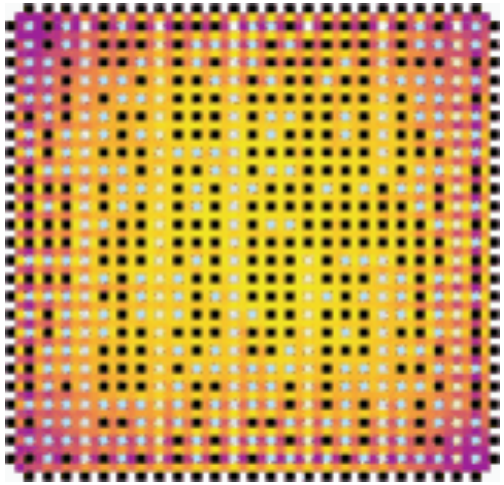
Placement (input)



Predicted



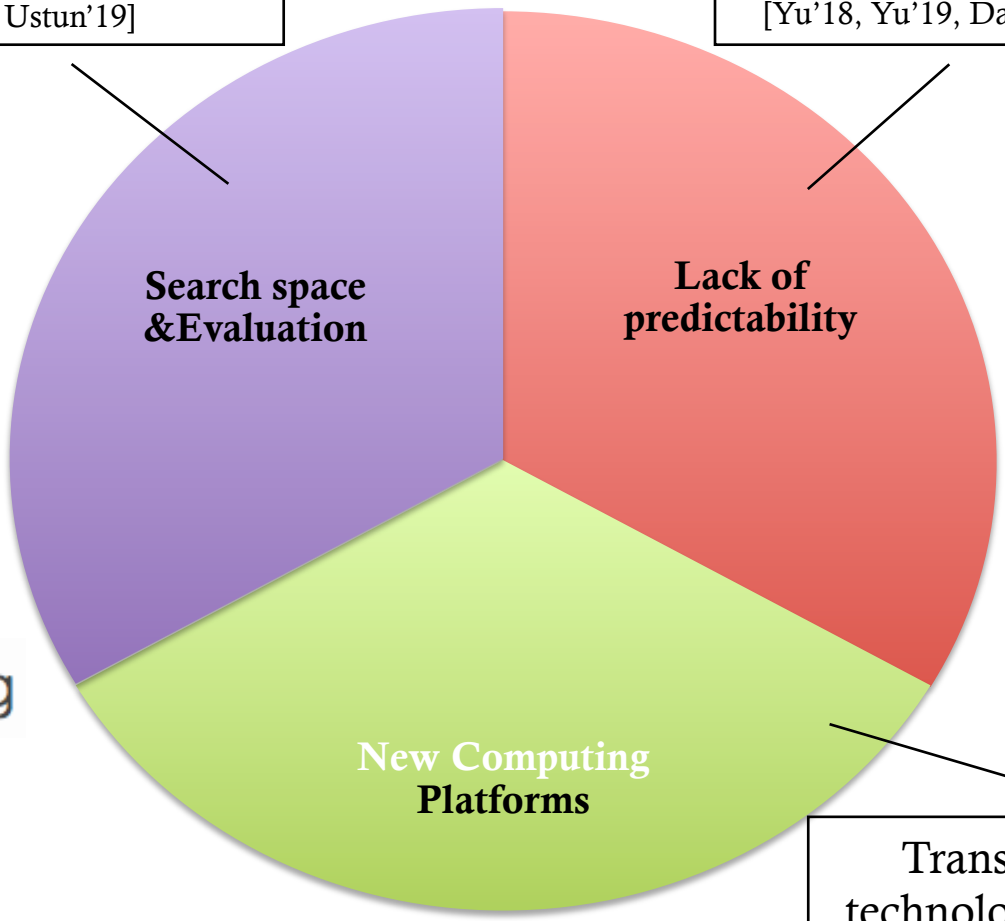
Ground truth



Conclusions

Online training; multi-stage exploration;
[Yu'18, Yu'19, Ustun'19]

Accurate estimation models; graph learning
[Yu'18, Yu'19, Dai'18, Zhou'19]



Transfer learning – technologies and designs & **open-source**

Verilog to Routing

VIVADO™

Quartus® Prime Design Suite

Accelerating Deep Learning Hardware Development with Cross-stack Co-Design Approach

Cornell University

Specialized DNN Processors are Ubiquitous

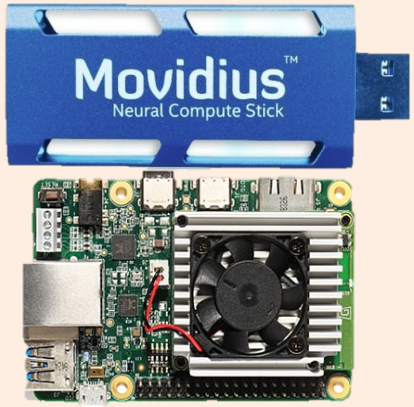
Mobile

Apple (A12)
Samsung (Exynos 9820)
Huawei (Kirin 970)
Qualcomm (Hexagon DSP)

Cloud

NVIDIA (cuDNN, CUDA-X AI)
Google (TPU)
Microsoft (Brainwave)
Xilinx (UltraScale + on EC2 F1)
Intel (Nervana)

Embedded

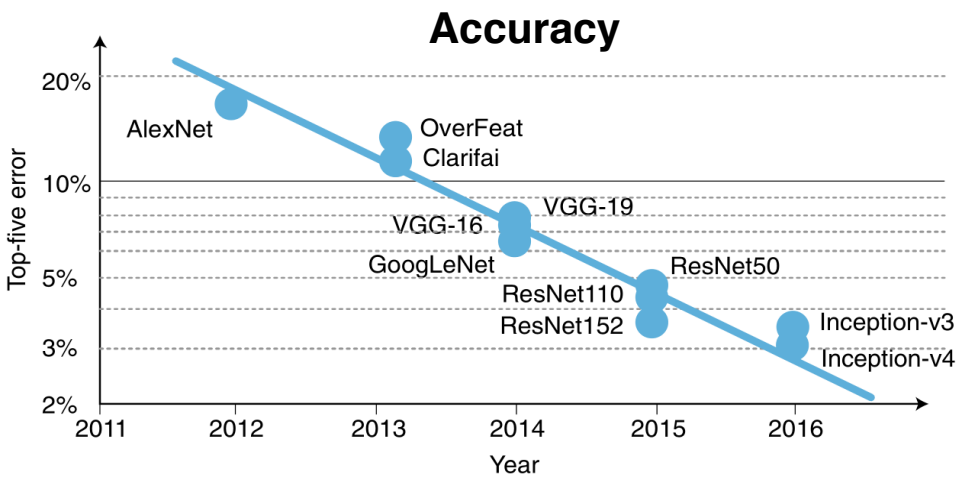


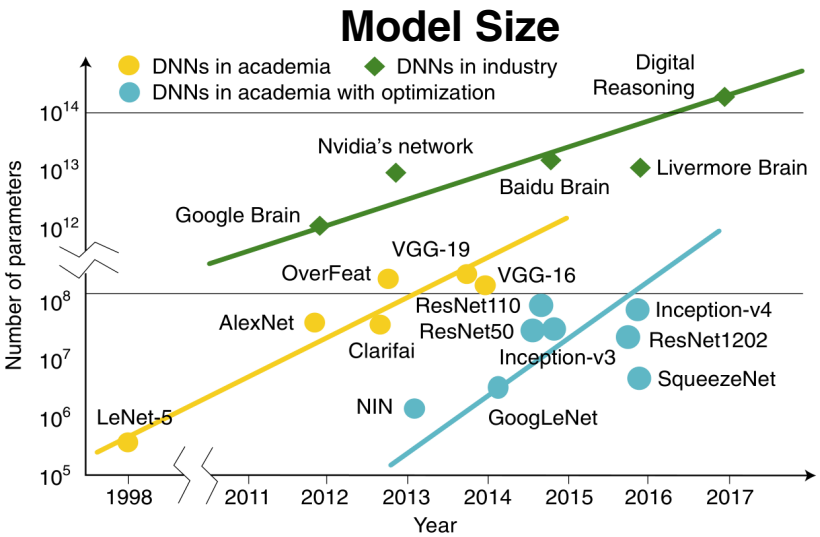
Google (Edge TPU)
Intel (Movidius)
Deephi/Xilinx (Zynq)
ARM (announced)

DNNs are Getting Larger and Deeper

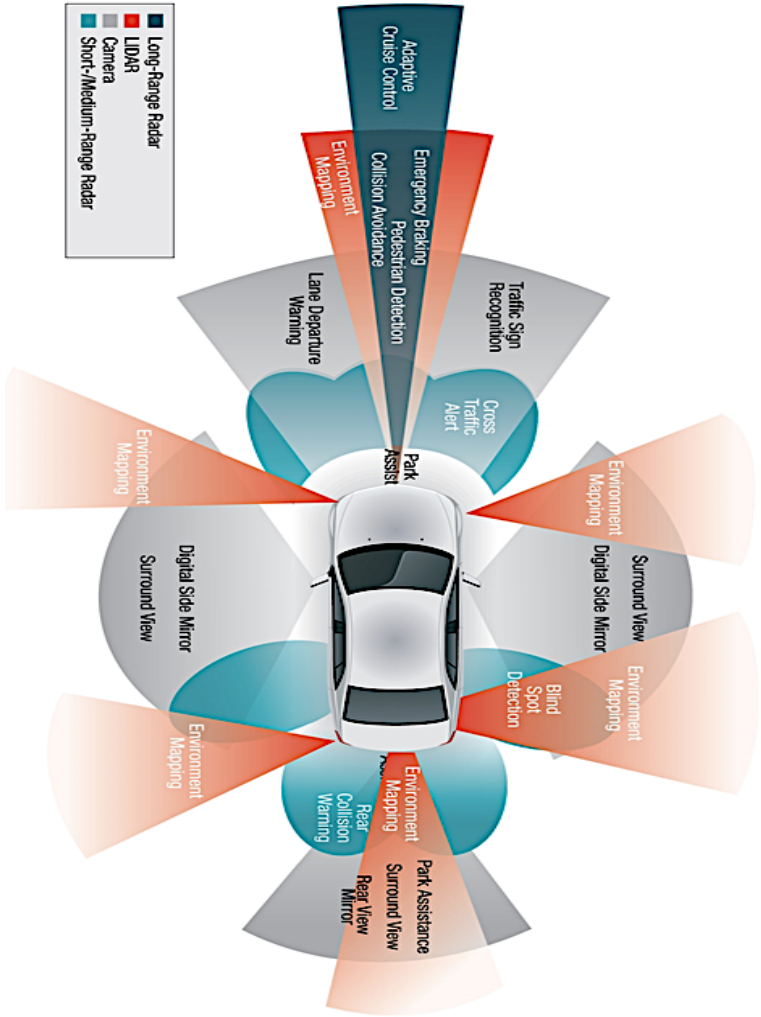
- DNNs are getting larger in both academia and industry
  - Increasing in compute and memory

- Challenges in DNN acceleration
  - Speed, size, power, heat, energy, etc.
  - Flexibility, cost, time-to-market, debugging/testing, etc.





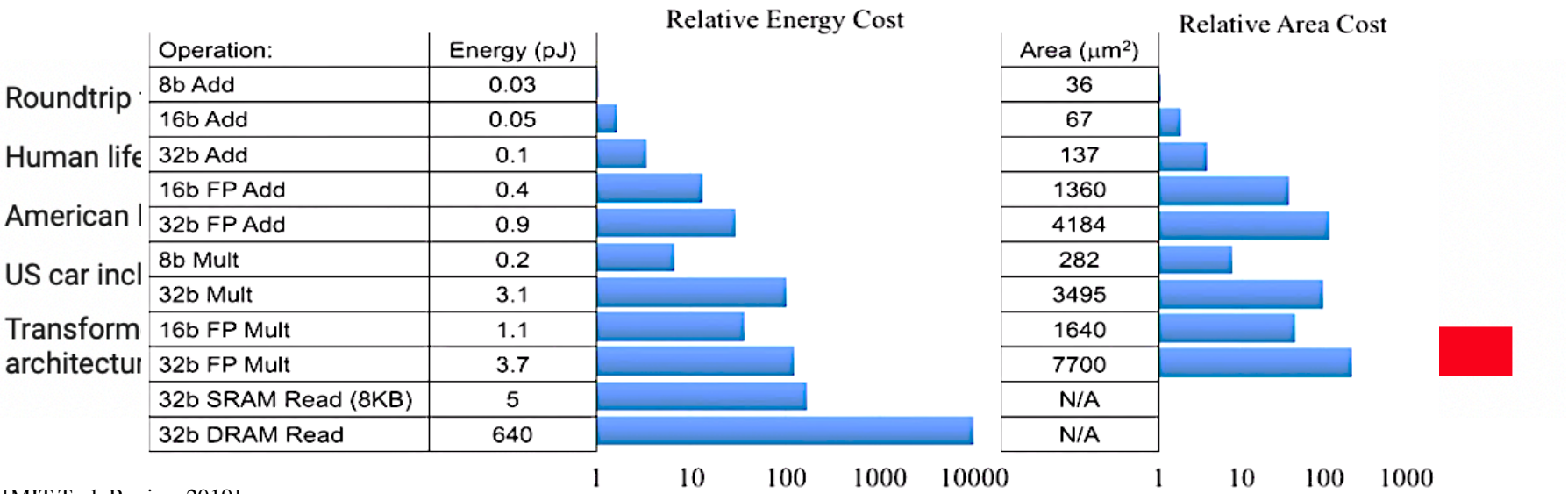
How Much Compute?



- ResNet-50 inferencing
  - 25 Million Weights
  - 300 Gops for HD Image
  - 9.4 Tops for 30fps
  - 12 Cameras, 3 nets = 338 Tops
- Training
  - 30Tops x 10^8 (train set) x 10^2 (epochs) = 10^{23} Ops

[Bill Dally, SysML 2018]

HW Cost for DNN Operations



[MIT Tech Review 2019]

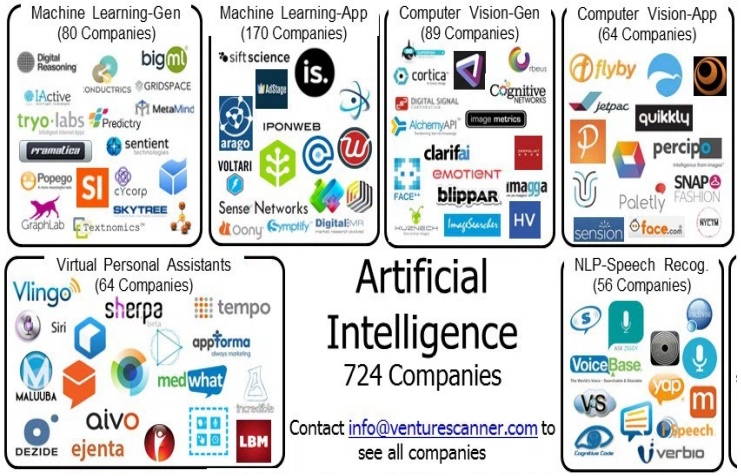
Instruction	Data	Type	Energy	Energy/Op
1 Op/Instr (*,+)	Memory	fp32	89 nJ	693 pJ
128 Ops/Instr (AX+B)	Memory	fp32	72 nJ	562 pJ
128 Ops/Instr (AX+B)	Cache	fp32	0.87 nJ	6.82 pJ
128 Ops/Instr (AX+B)	Cache	fp16	0.47 nJ	3.68 pJ

[Horowitz ISSCC 2014]

ML Hardware Compilation

ML Hardware Providers

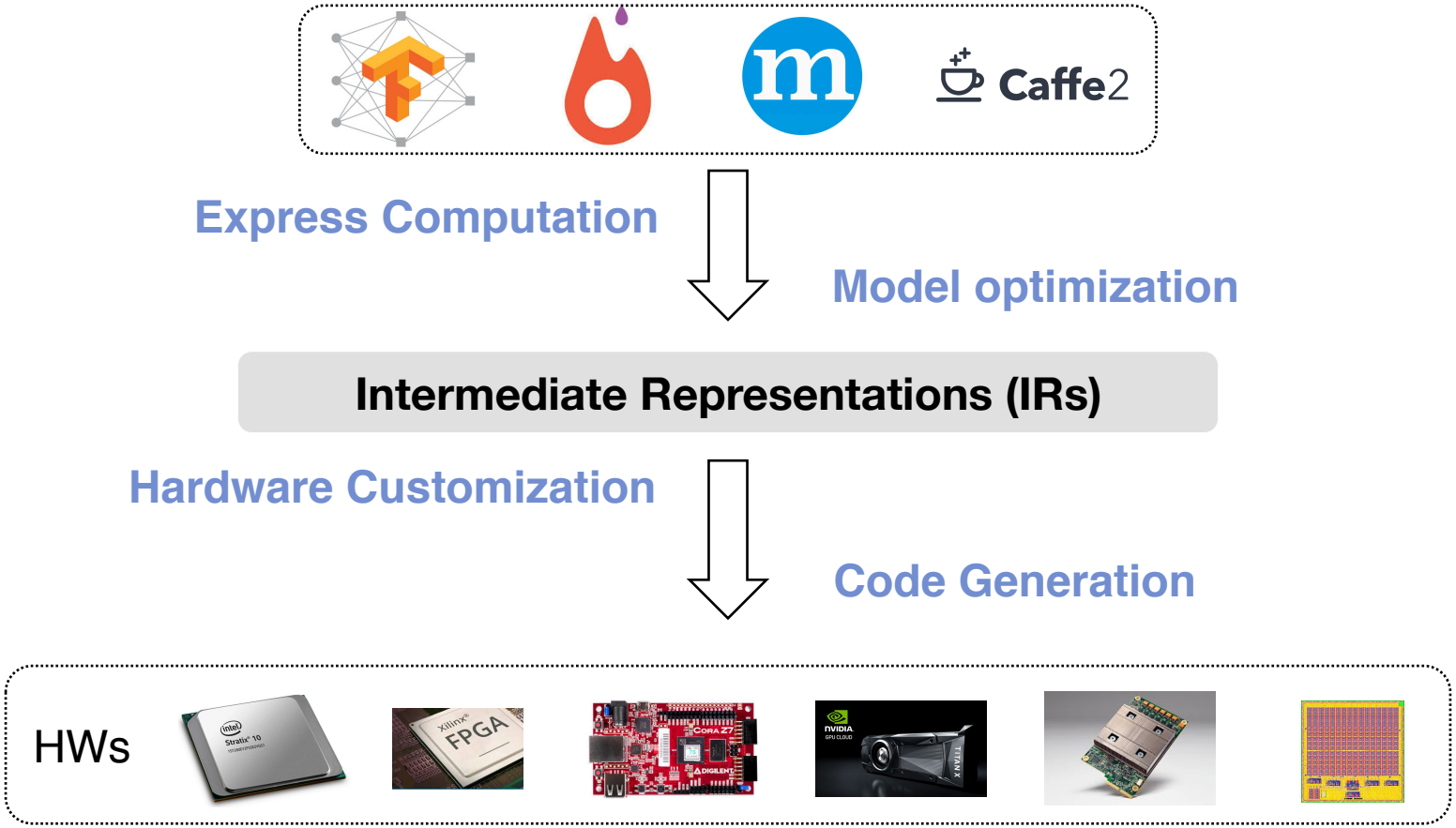
Trained Model



ML Customers

ML Hardware Compilation

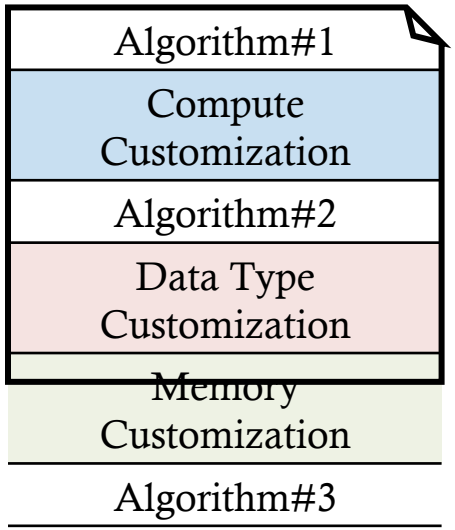
- End-to-end compilation flow



Hardware Customization in High-Level Synthesis

- Driving example: convolutional kernel

```
for (int y = 0; y < N; y++)
  for (int x = 0; x < N; x++)
    for (int r = 0; r < 3; r++)
      for (int c = 0; c < 3; c++)
        out[x, y] += image[x+r, y+c] * kernel[r, c]
```



Entangled hardware customization and algorithm

- Less portable
- Less maintainable
- Less productive

```
#pragma HLS array_partition variable=filter dim=0
  hls::LineBuffer<3, N, ap_fixed<8,4> > buf;
  hls::Window<3, 3, ap_fixed<8,4> > window;
  for(int y = 0; y < N; y++) {
    for(int xo = 0; xo < N/M; xo++) {
#pragma HLS pipeline II=1
      for(int xi = 0; xi < M; xi++) {
        int x = xo*M + xi;
        ap_fixed<8,4> acc = 0;
        ap_fixed<8,4> in = image[y][x];
        buf.shift_up(x);
        buf.insert_top(in, x);
        window.shift_left();
        for(int r = 0; r < 2; r++)
          window.insert(buf.getval(r,x), i, 2);
        window.insert(in, 2, 2);
        if (y >= 2 && x >= 2) {
          for(int r = 0; r < 3; r++) {
            for(int c = 0; c < 3; c++) {
              acc += window.getval(r,c) * kernel[r][c];
          }}
          out[y-2][x-2] = acc;
}}}}
```

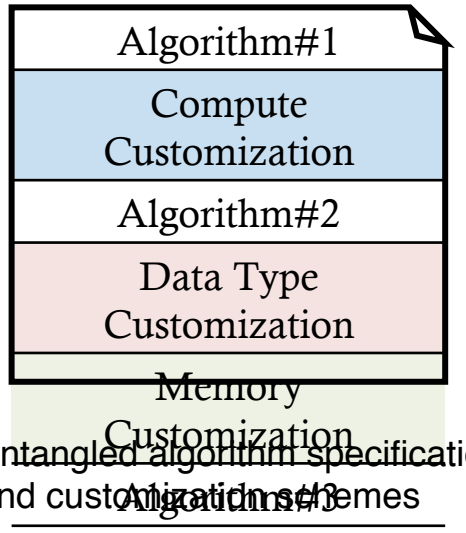
Custom compute (Loop tiling)

Custom data type (Quantization)

Custom memory (Reuse buffers)

Decoupling Algorithm from Hardware Customization

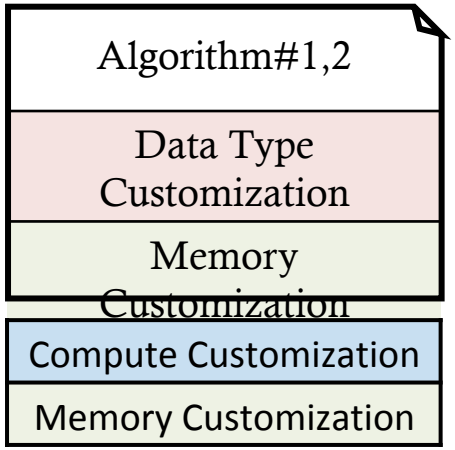
HLS C



Entangled algorithm specification and customization schemes

Intel HLS
Xilinx Vivado HLS
Canis, et al. FPGA'11

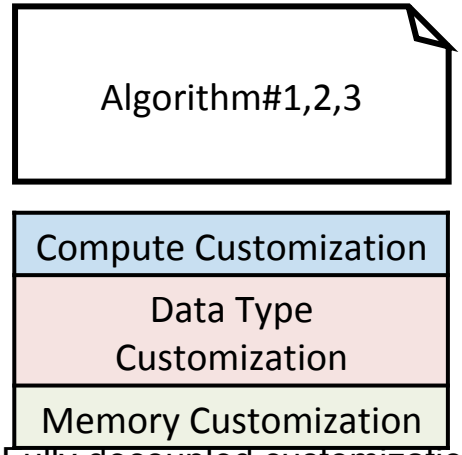
Halide, TVM, etc.



Decoupled temporal schedules

Ragan-Kelly, et al. SIGPLAN'13
Baghdadi, et al. arXiv'18
Rong, et al. arXiv'17
Pu, et al. TACO'17
Chen, et al. OSDI'18

HeteroCL



Fully decoupled customization schemes +
Clean abstraction capturing the interdependence

Lai, et al., FPGA'19 (BPA)

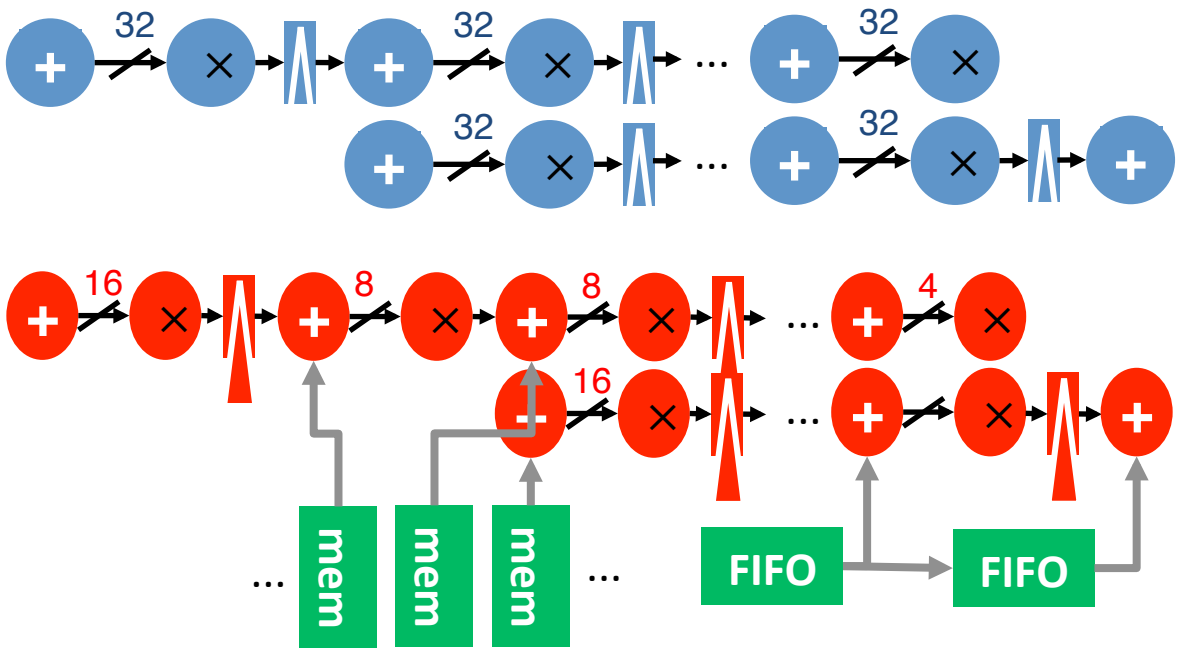
http://halide-lang.org
https://github.com/dmlc/tvm/ (http://tvm.ai)
http://heterocl.csl.cornell.edu/web/index.html

Essential Techniques for Hardware Acceleration

Compute customization
- Parallelization
- Pipelining, etc.

Data type customization
- Low-bitwidth integer
- Fixed point, etc.

Memory customization
- Banking
- Data reuse, etc

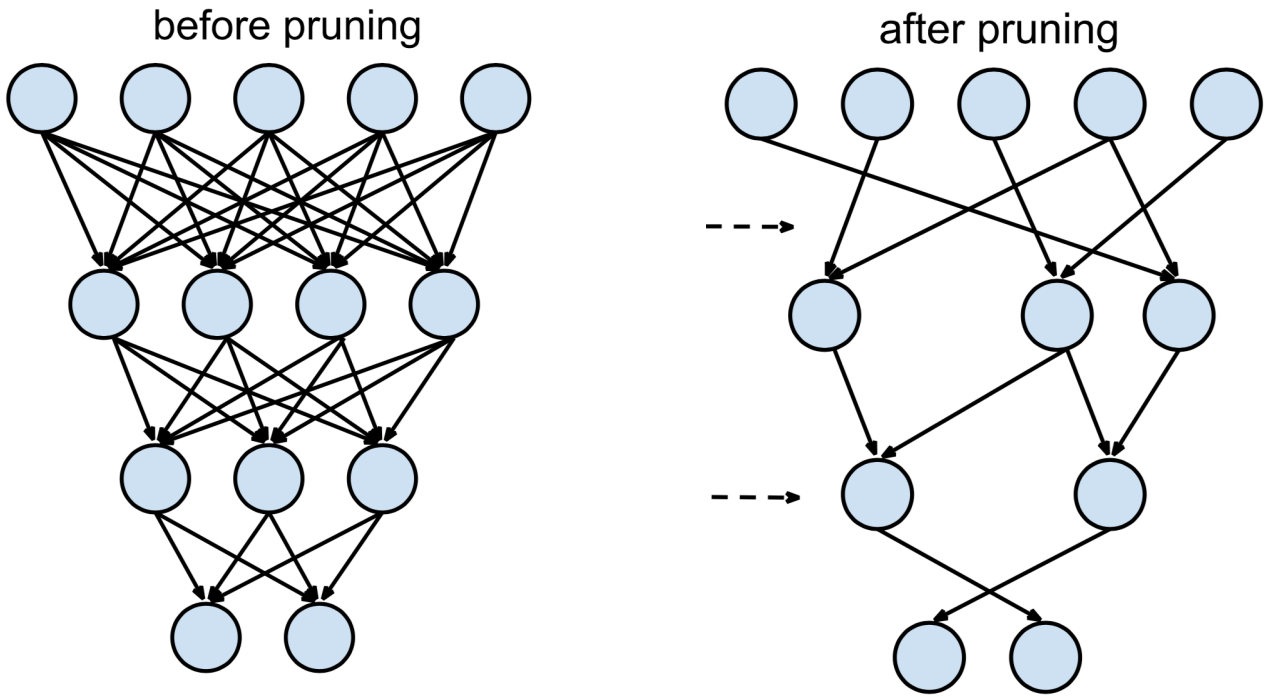


There exists interdependence among different customizations

HeteroCL: A Multi-Paradigm Programming Infrastructure for Software-Defined Reconfigurable Computing. Lai et. al., FPGA'19
http://heterocl.csl.cornell.edu/web/index.htmL

Model Optimization – *Pruning*

- Example: pruning neural networks



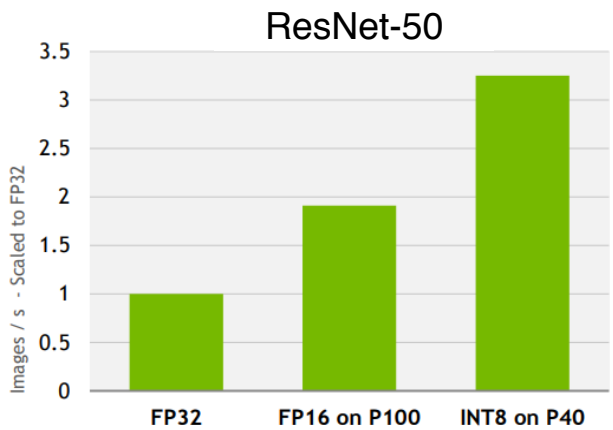
Han, Song, et al. "Learning both weights and connections for efficient neural network." Advances in neural information processing systems. *NeurIPS* 2015

Model Optimization – *Quantization*

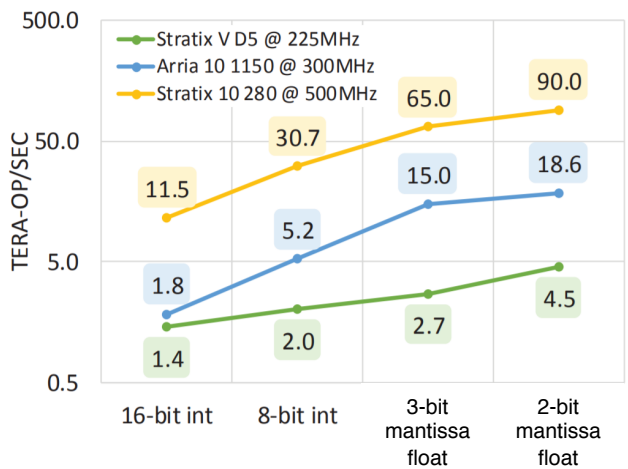
- Example: quantization is a key for efficiency
  - Reduced precision
    - narrower arithmetic operators
    - less area and energy/op
    - fewer bits of storage per data

Critical for compute & memory

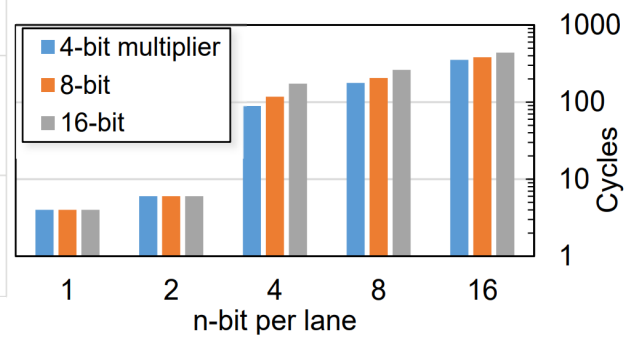
GPU Performance



FPGA Performance



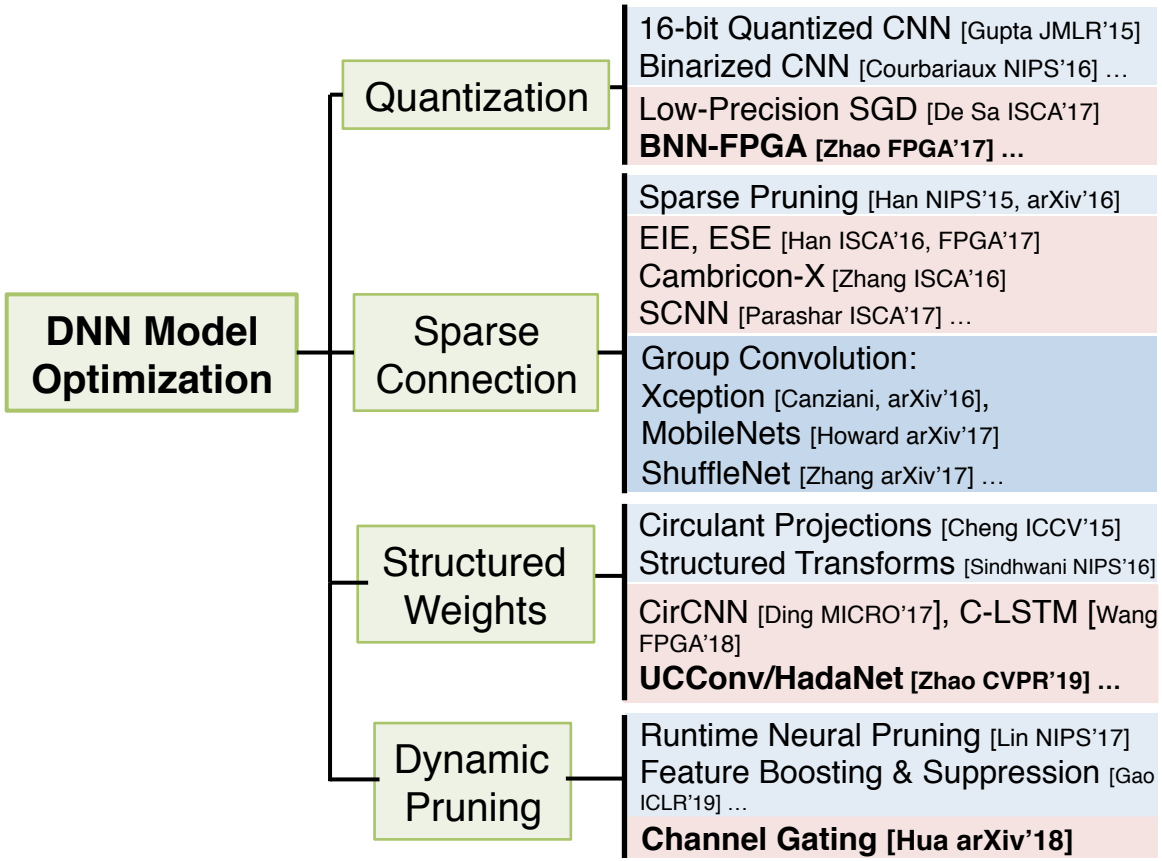
In-Memory Accelerator Performance (DRISA)



E. Chung, J. Fowers et al. Serving DNNs in Real Time at Datacenter Scale with Project Brainwave, *IEEE Micro*, April 2018.

S. Li, D. Niu, K.T. Malladi, H. Zheng, B. Brennan Y. Xie, DRISA: A DRAM-based Reconfigurable In-Situ Accelerator. *Micro'18*, Oct 2018.

Spectrum of DNN Compression Techniques



Goal: Enabling efficient inference in resource-limited devices in mobile or edge settings

Co-design of algorithms & hardware yields highest efficiency

Decoupled Compute Customization

HeteroCL code

Algorithm

```
import hetrocl as hcl
r = hcl.reduce_axis(0, 3)
c = hcl.reduce_axis(0, 3)
out = hcl.compute(N, N),
    lambda y, x:
        hcl.sum(image[x+r, y+c]*kernel[r, c],
            axis=[r, c]))
```

Declarative programming

Decoupled customization

```
s = hcl.create_schedule()
xo, xi = s[out].split(out.x, factor=M)
s[out].reorder(xi, xo, out.y)
```

Customization primitives

- More productive / less labor-intensive

HLS code

```
for (int y = 0; y < N; y++)
  for (int x = 0; x < N; x++)
    for (int r = 0; r < 3; r++)
      for (int c = 0; c < 3; c++)
        out[x, y] += image[x+r, y+c] * kernel[r,
c]

for (int xi = 0; xi < M; xi++)
  for (int xo = 0; xo < N/M; xo++)
    for (int y = 0; y < N; y++)
      for (int r = 0; r < 3; r++)
        for (int c = 0; c < 3; c++)
          out[xi+xo*M, y] +=
            image[xi+xo*M+r, y+c] * kernel[r, c]
```

Tile loop

Reorder loops

Optimized BNN in HLS-C

```
template<int M, int N, int I, int L>
void conv(ap_int<32> input[MAX_FMAP_PACK_SIZE],
          ap_int<32> output[MAX_FMAP_PACK_SIZE],
          const ap_int<8> threshold[MAX_FMAP],
          hls::LineBuffer<F, I, bit> buf[M]) {
  int O = I - F + 1, ifmap_size = I * I, ofmap_size = O * O;
  hls::Window<F, F, bit> window[M];
  for (int y = 0; y < O; y++) {
    for (int m = 0; m < M; m++) {
    #pragma HLS pipeline
      for( int x = 0; x < F - 1; x++) {
        int i_index = x + (y + F - 1) * I + m * ifmap_size;
        bit newBit = GET_BIT(input, i_index, PACK_WIDTH_LOG);
        fillBuffer<F, I>(window[m], buf[m], x, newBit);
    }}
    for (int x = 0; x < O; x++) {
      for (int m = 0; m < M; m++) {
        int i_index = x + F - 1 + (y + F - 1) * I + m * ifmap_size;
        bit newBit = GET_BIT(input, i_index, PACK_WIDTH_LOG);
        fillBuffer<F, I>(window[m], buf[m], x + F - 1, newBit);
      }
      for (int n = 0; n < N; n++) {
      #pragma HLS pipeline
        int sum = 0;
        int o_index = x + y * O + n * ofmap_size;
        for (int m = 0; m < M; m++) {
          int one_out = 0, mac_num = 0;
          for (int c = 0; c < F; c++) {
            for (int r = 0; r < F; r++) {
              if (if_mac(x + c, y + r, I)) { //neglect padding pixels in mac
                int i_index = x + c + (y + r) * I + m * ifmap_size;
                int w_index = c + r * F + (n + m * N) * FILTER_SIZE;
                if (L == 0) one_out += window[m].getval(r, c) == w_conv1[w_index];
                else        one_out += window[m].getval(r, c) == w_conv2[w_index];
                mac_num++;
          }}}
          sum += (one_out << 1) - mac_num;
        }
        SET_BIT(output, o_index, PACK_WIDTH_LOG, sum > threshold[o_index] ? 1 : 0);
}}}}
```

Applied customization techniques

- **Compute:** tiling, pipelining, reordering
- **Data type:** bit packing
- **Memory:** partitioning, line buffer, window buffer

Compute customization

Data type customization

Memory customization

HeteroCL Supported Customization Primitives

Compute customization

Primitive	Description
	Loop transformation
`C.split(i, v)`	Split loop `i` of operation `C` into a two-level nest loop with `v` as the factor of the inner loop.
`C.fuse(i, j)`	Fuse two sub-loops `i` and `j` of operation `C` in the same nest loop into one.
`C.reorder(i, j)`	Switch the order of sub-loops `i` and `j` of operation `C` in the same nest loop.
`P.compute_at(C, i)`	Merge loop `i` of the operation `P` to the corresponding loop level in operation `C`.
	Parallelization
`C.unroll(i, v)`	Unroll loop `i` of operation `C` by factor `v`.
`C.parallel(i)`	Schedule loop `i` of operation `C` in parallel.
`C.pipeline(i, v)`	Schedule loop `i` of operation `C` in pipeline manner with a target initiation interval `v`.

Memory customization

Primitive	Description
`C.partition(i, v)`	Partition dimension `i` of tensor `C` with a factor `v`.
`C.reshape(i, v)`	Pack dimension `i` of tensor `C` into words with a factor `v`.
`memmap(t, m)`	Map a list of tensors `t` with mode `m` to new tensors. The mode `m` can be either vertical or horizontal.
`P.reuse_at(C, i)`	Create a reuse buffer storing the values of tensor `P`, where the values are reused at dimension `i` of operation `C`.

Data type customization

Primitive	Description
`quantize(t, d)`	Quantize a list of tensors `t` from floating to fixed point type `d` in the format defined in Table 2.
`downsize(t, d)`	Downsize a list of tensors `t` from integers with larger bitwidth to integers `d` with smaller bitwidth in the format defined in Table 2.

Optimized BNN in HeteroCL

- Development time: < 3 days
- Final speedup: 63x

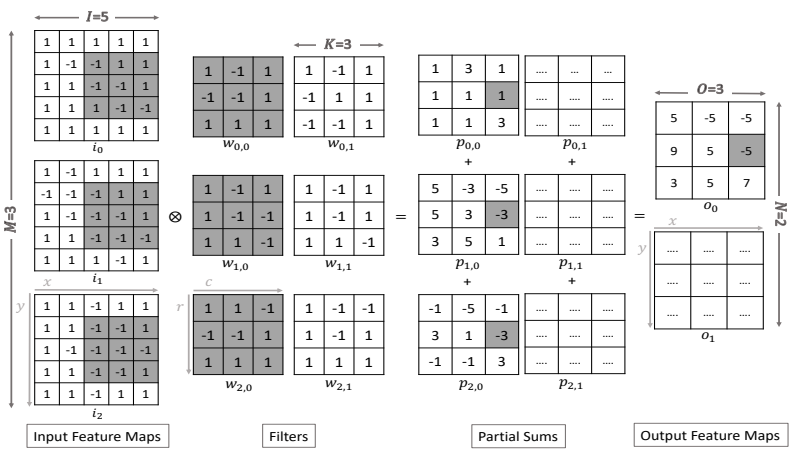
```
import heteroCL as hcl
rc = hcl.reduce_axis(0, in_fmaps)
ry = hcl.reduce_axis(0, F)
rx = hcl.reduce_axis(0, F)
C = hcl.compute((1, out_fmaps, O, O),
  lambda nn, ff, yy, xx:
  hcl.select(
      hcl.sum(A[nn,rc,yy+ry,xx+rx] * B[ff,rc,ry,rx], axis=[rc,ry,rx]) >
            threshold[nn,ff,yy,xx], 1, 0 ),
  dtype=hcl.UInt(1)) # binarize the data type 0,1 only

s.quantize(C, hcl.UInt(32))
s[C].split(C.axis[1], factor=5)
s[C].unroll(C.axis[2], factor=5)
s[C].pipeline(C.axis[3])
lb = s[A].reuse_at(C, C.axis[0])
wb = s[lb].reuse_at(C, C.axis[1])
```

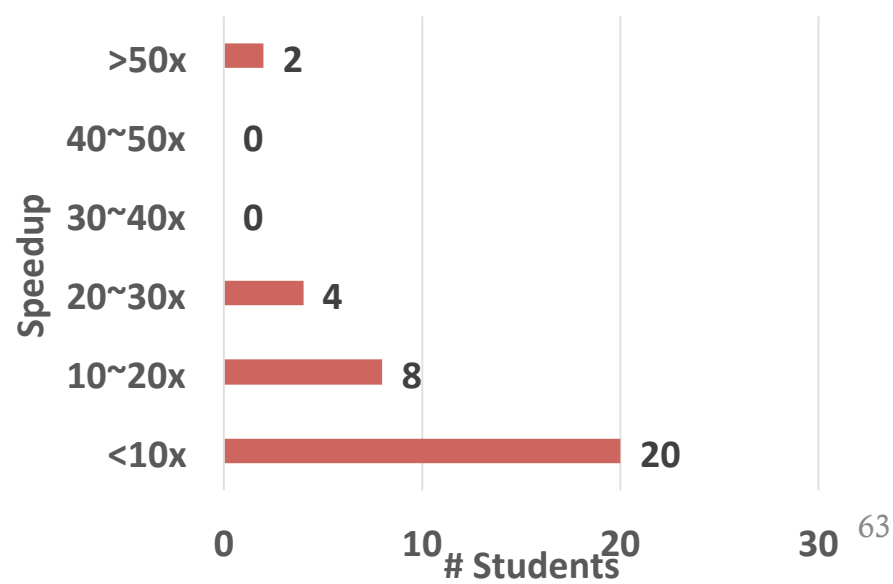
✓ **More productive**

✓ **More maintainable**

Case Study: Binarized Neural Network (BNN)

- Cornell graduate course study
  - 34 students: graduates and senior undergrads
- Goal: higher speedup => higher score
  - Baseline: unoptimized BNN on ARM (Zynq)
  - Time: two weeks





[1] https://www.csl.cornell.edu/courses/ece5775/

HeteroCL - ResNet18

Basic ops

```
def relu(x, name="relu"):
    return hcl.compute(x.shape, lambda *args: hcl.select(x[args] > 0.0, x[args],
            0.0), name, attrs=OrderedDict([('app_name', tvm.make.StringImm('relu'))]))
def softmax(out, x):
    … …
    max_elem = hcl.compute((m, ), lambda i: max(x[i, k], axis=k))
    expsum = hcl.compute((m, ), lambda i: sum(tvm.exp(x[i, k] - max_elem[i]), axis=k))
    return hcl.update( out, lambda i, j: tvm.exp(x[i, j] - max_elem[i]) / expsum[i])
```

HeteroCL(Python)
~400 lines

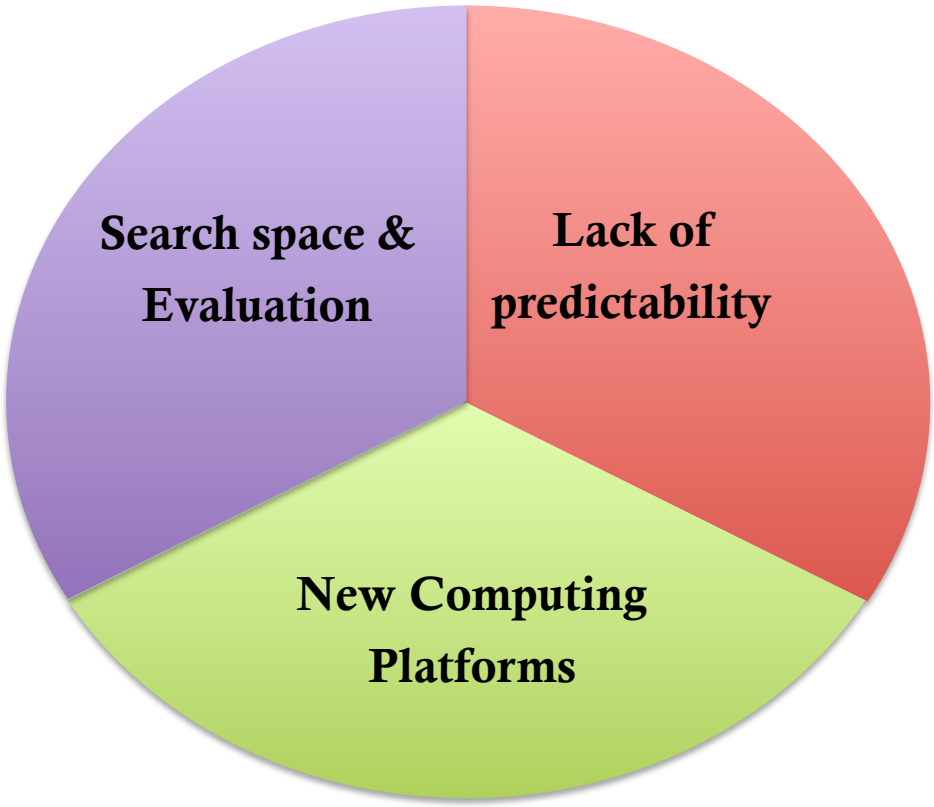
Layers

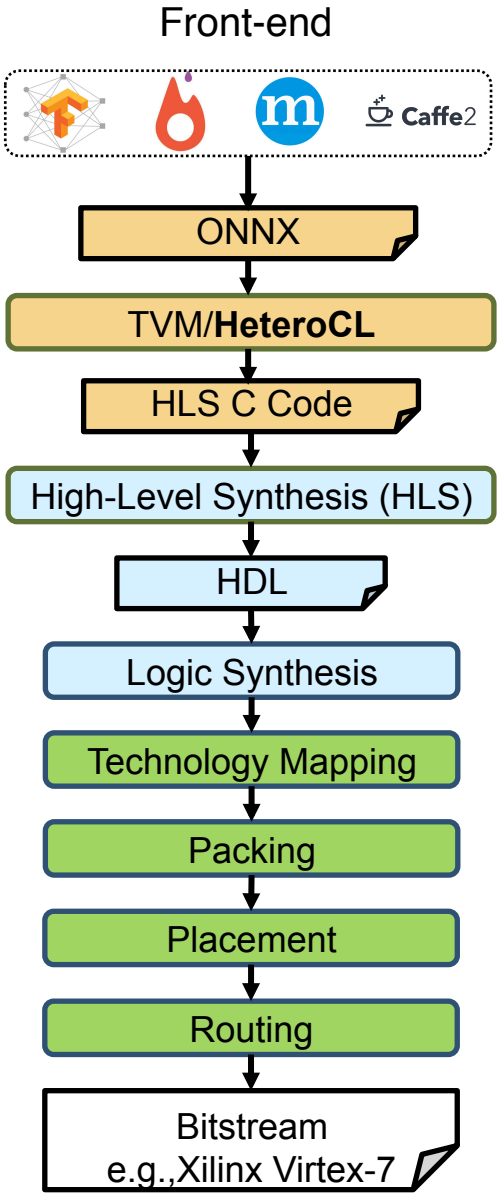
```
def dense(data, weight, bias=None, name="dense"):
    ... ...
    k = hcl.reduce_axis(0, in_dim)
    attrs=OrderedDict([('k', in_dim),('j', out_dim),
        ('i', batch),('app_name', tvm.make.StringImm('mm'))])
    return hcl.compute((batch, out_dim), lambda i, j: sum(data[i, k] * weight[j, k], axis=k),
name, attrs=attrs)

def max_pool(data, kernel, stride, padding=[[0,0],[0,0]], name="max_pool"):
    #maxpool
    return hcl.compute(...)
def batchnorm(x, beta, gamma, mean, var, name="batchnorm"):
    #BN
    return hcl.compute(...)
def conv2d_nchw(Input, Filter, name="conv2d", stride=[1,1], padding=[[0,0],[0,0]]):
    #conv2d
    return hcl.compute(...)
```

Build and Compile
w Xilinx VivadoHLS

```
# define default schedules …
scheme_new = hcl.create_scheme(arg_list, build_resnet)
s_new = hcl.create_schedule_from_scheme(scheme_new)
f_new = hcl.build(s_new, target="vhls")
res = feedback(vivado_hls('dut_csynth.xml'), 'min')
```

Compilation Infrastructure





- Software Hardware Co-design
  - Model accuracy vs. HW cost
  - Lack of predictability
    - Mis-correlation between DL algorithm & HW
  - A large number of NN model variations

In-box ML-assisted Autotuning – User Interface

➢ original code

```
#p0 = 5 #factor
#p1 = 32 #int16, int32, etc
s.quantize(C, hcl.UInt(32))
s[C].split(C.axis[1], factor=5)
s[C].unroll(C.axis[2], factor=5)
f_new = hcl.build(s_new, target="vhls")
res = feedback(vivado_hls(.xml'), 'min')
```

➢ code for autotuning

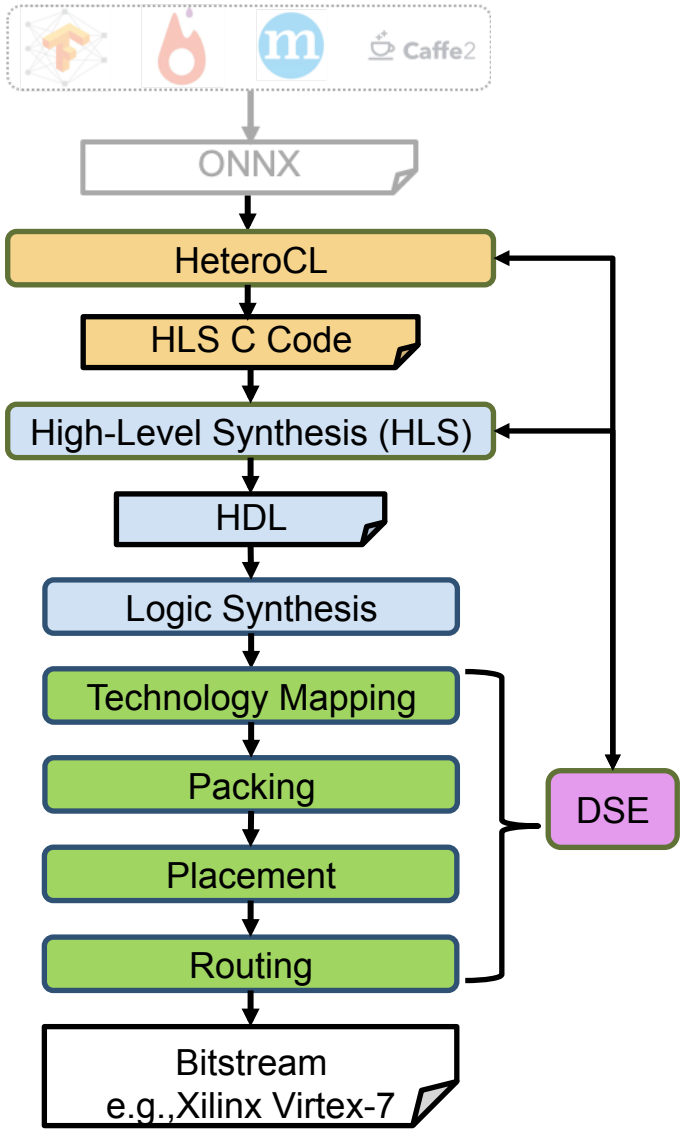
```
import uptune.tune as autotune
p0 = autotune(5,(2,3,4,5))
p1 = autotune(hcl.UInt(32),(hcl.UInt(16), hcl.UInt(8)))
s.quantize(C, hcl.UInt(p0))
s[C].split(C.axis[1], factor=5)
s[C].unroll(C.axis[2], factor=5)
f_new = hcl.build(s_new, target="vhls")
res = feedback(vivado_hls(.xml'), 'min’)
@uptune.feedback(‘min’)
        fps = get_fps(res)
```

*autotune(default, range, name, *args)*

Parameters:

- **default** (*str / real-value*) – the default value with tuning disabled.
- **range** (*list / tuple*) – the tuning range (for real-value and enumeration)
- **name** (*str, optional*) – name for tracking variable tuning trend

In-box Learning-assisted Autotuning



Model Opt & HW Customization

```
import uptune.tune as autotune
p0 = autotune(5,(2,3,4,5))
p1 = autotune(16,(8,16,32))
s.quantize(C, hcl.UInt(p0))
s[C].split(C.axis[1], factor=p1)
s[C].unroll(C.axis[2], factor=p1)
f_new = hcl.build(s_new, target="vhls")
@uptune.feedback('max')
        acc = get_acc(res)
```

Synthesis

```
candidate = [b', rw', rs', rf', rw -z', rf -z']
syn = []
for i in range(0,24):
   t = autotune(candidate[r.randint(0,5)], candidate)
   syn.append(t)
```

PnR

```
auto_dsp_recognition = autotune('On', ['On', 'Off'])
disable_register_merging_across_hierarchies =
autotune('On', ['On', 'Off', 'Auto'])
synthesis_effort = autotune('Auto', ['Auto', 'Fast'])
timing_driven_synthesis = autotune('On', ['On', 'Off'])
@uptune.feedback('min')
        fps, acc = get_fps(res)
```

HetoroCL Results

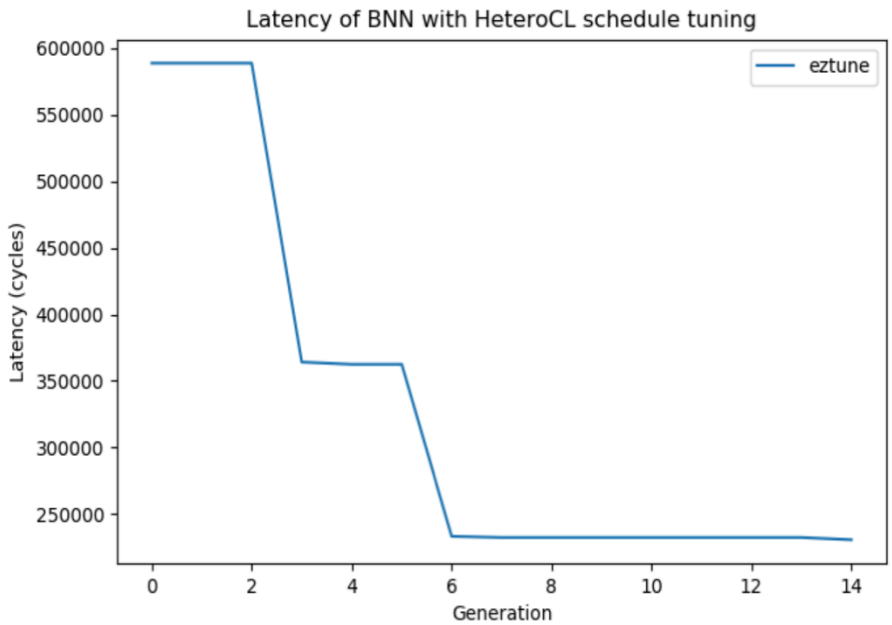
Benchmark	Application Field	Back end	Data type	Performance (GOPs)	Speedup
GEMM	Matrix multiplication	CPU (Intel MKL)	float32	76.0	1.0
		FPGA	float32	245.9	3.2
			fixed16	807.6	10.6
LeNet	Convolutional neural network	CPU (TVM TOPI)	float32	15.4	1.0
		FPGA	float32	79.8	5.2
			fixed16	137.8	8.9

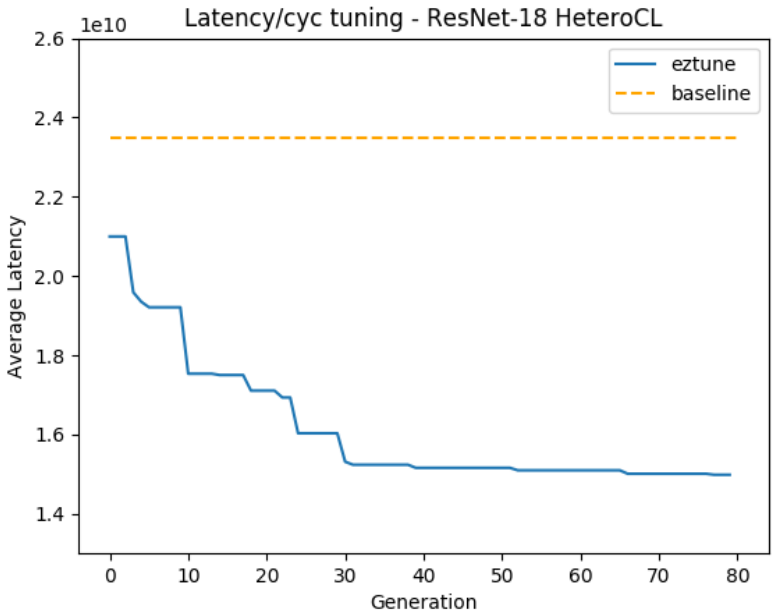
Benchmark	Application Field	Speedup
KNN Digit Recognition	Image classification	12.5
K-Means	Clustering	16.0
Smith-Waterman	Genomic sequencing	20.9

HeteroCL: A Multi-Paradigm Programming Infrastructure for Software-Defined Reconfigurable Computing. Lai et. al., FPGA'19
http://heterocl.csl.cornell.edu/web/index.htmL

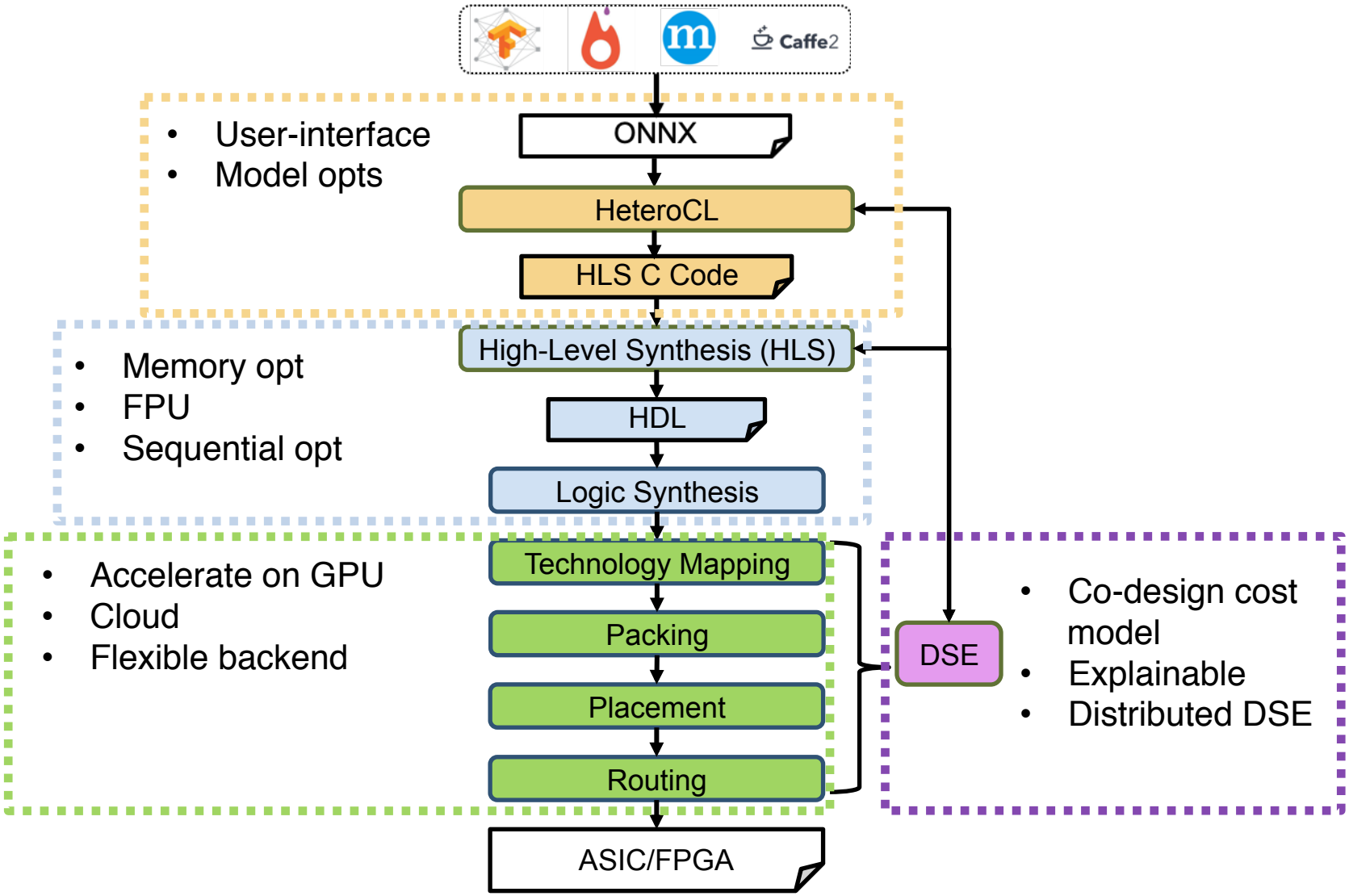
Preliminary DSE Results

- AWS f1 instance





Our Scope


Caffe2
ONNX
HeteroCL
HLS C Code
High-Level Synthesis (HLS)
HDL
Logic Synthesis
Technology Mapping
Packing
Placement
Routing
ASIC/FPGA
DSE
• User-interface
• Model opts
• Memory opt
• FPU
• Sequential opt
• Accelerate on GPU
• Cloud
• Flexible backend
• Co-design cost model
• Explainable
• Distributed DSE

Thank you!

Cunxi Yu
University of Utah