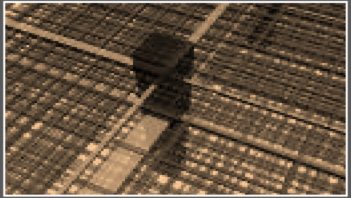
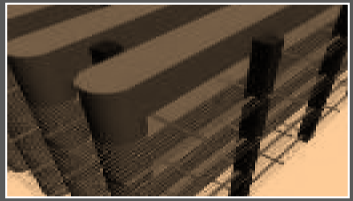
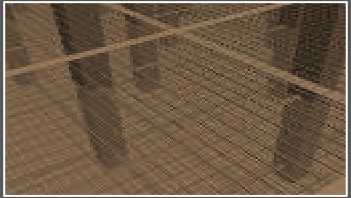
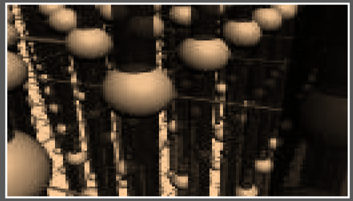
Machine Learning-Powered VLSI Physical Design Automation

Sung Kyu Lim
Georgia Tech / DARPA

EDPS 2023, Lunch Keynote Talk

10/5/2023

Agenda

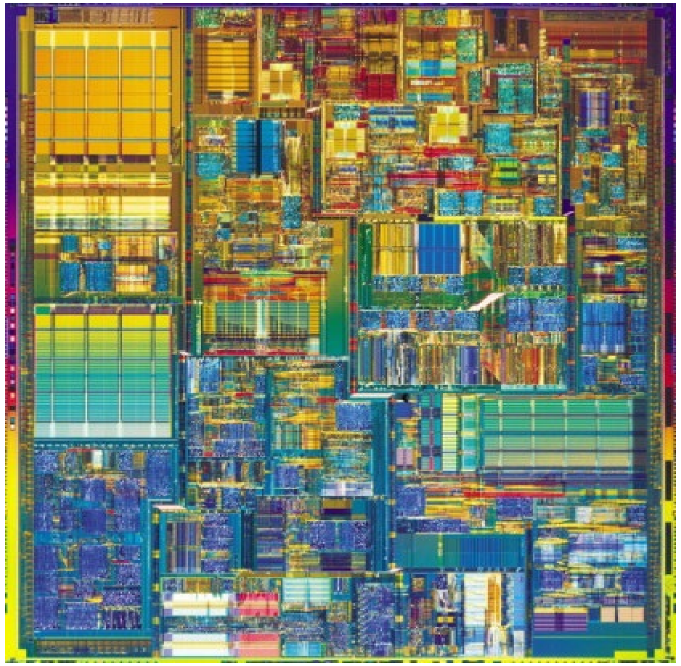
- **ML-outside**
  - **ML-Powered VLSI Clock Routing**
- **ML-inside**
  - **ML-Powered Timing Optimization [DAC 2023 Best Paper Award]**
- **ML-inside**
  - **ML-Powered VLSI Circuit Placement (if time permits)**

Machine Learning EDA: ML Outside

- Sea of knobs
  - PPA and TAT depend heavily on how these are tuned

EDA tool knobs

Human
vs
AI

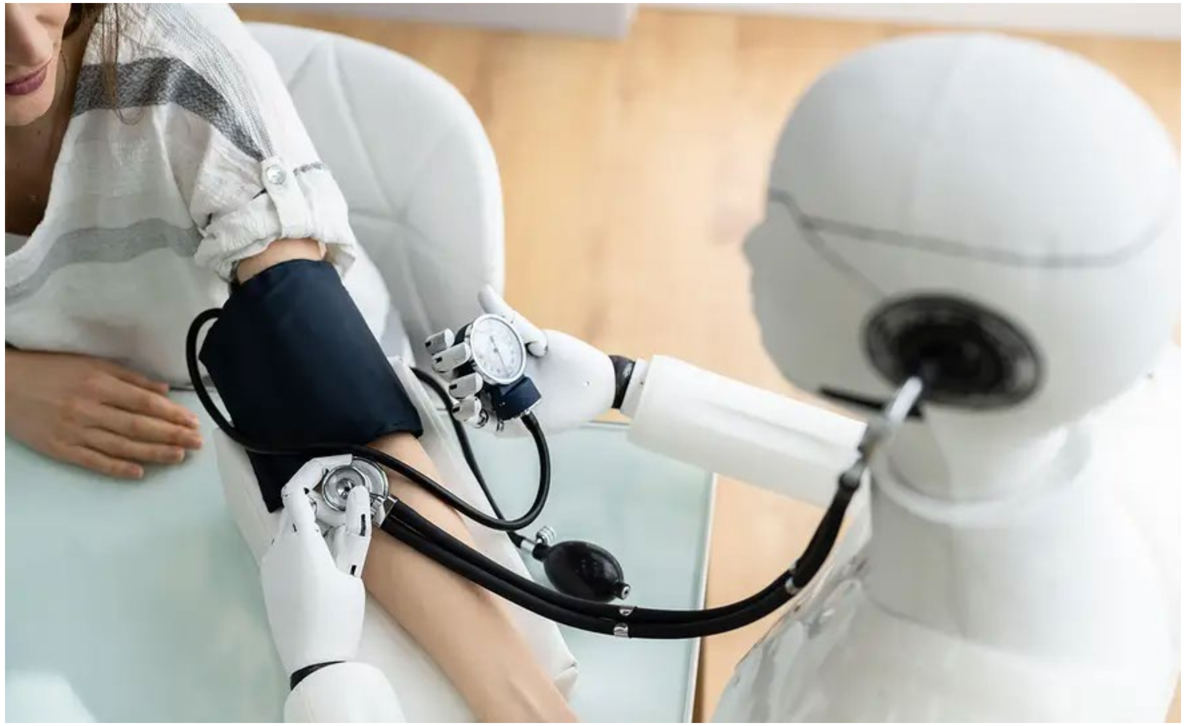


PPA and TAT

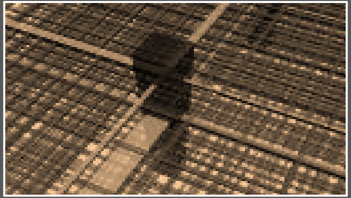
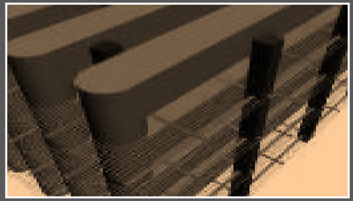
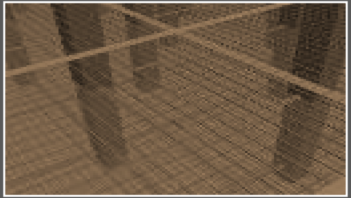
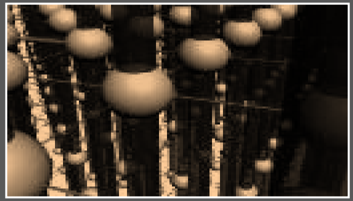
Machine Learning EDA: ML Inside

- **Physical design is harder and harder**
  - **Can AI help them perform better (or fix problems)?**

netlist
↓
floorplanning + PDN routing
↓
placement + pre-CTS opt
↓
clock routing + post-CTS opt
↓
signal routing + post-route opt
↓
PPA and reliability



ML-Powered Clock Routing

Two Clock Trees

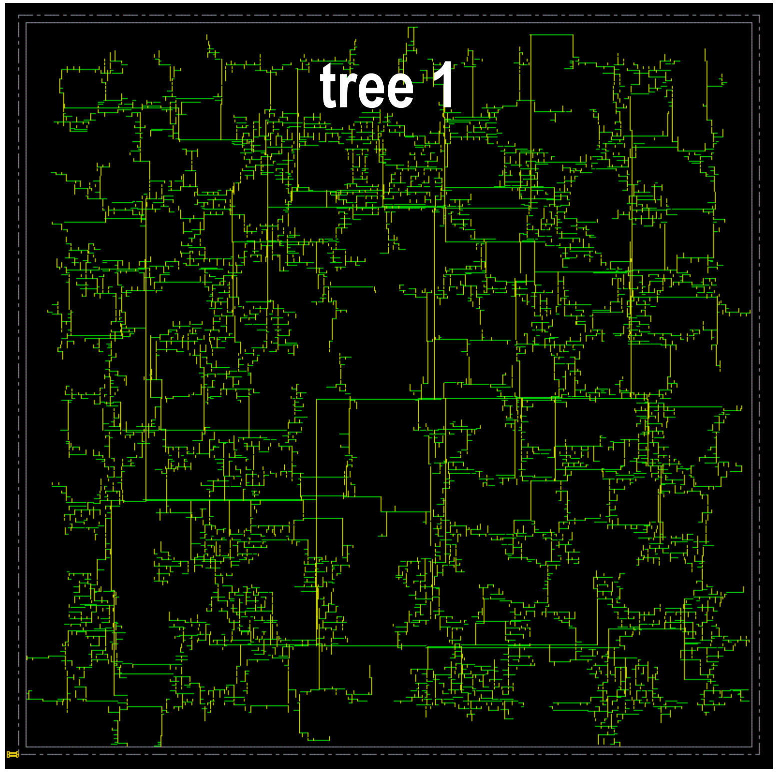
knob	value
Target skew	0.13ns
Max fanout	195
Max cap (trunk)	0.04pF
Max cap (leaf)	0.10pF
Target slew (trunk)	0.23ns
Target slew (leaf)	0.26ns
Target latency	0.4ns
eGR metal usage	1, 2, 3, 4
Cell density	0.6

Clock tree 1

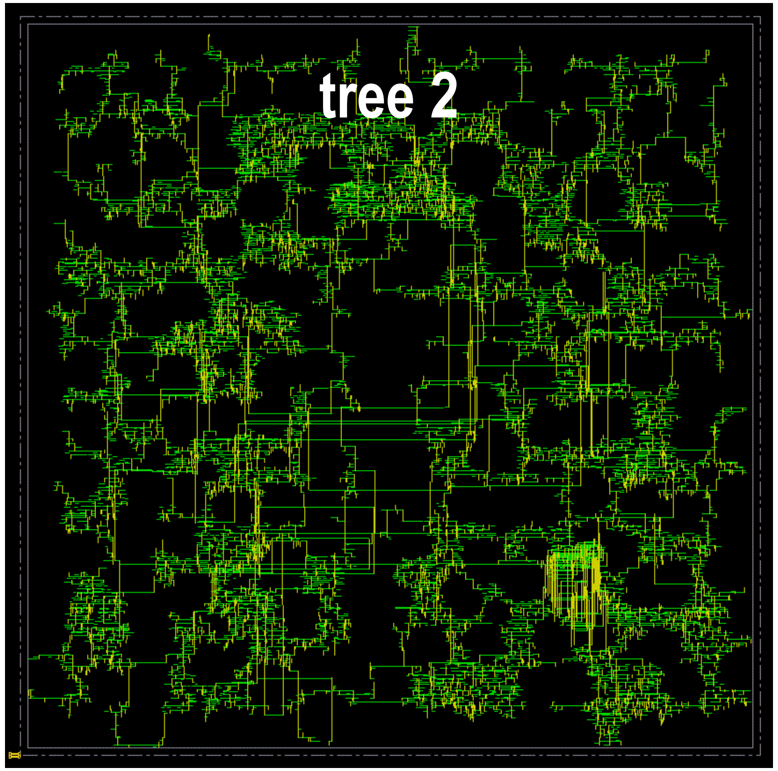
knob	value
Target skew	0.08ns
Max fanout	175
Max cap (trunk)	0.03pF
Max cap (leaf)	0.07pF
Target slew (trunk)	0.21ns
Target slew (leaf)	0.03ns
Target latency	0.2ns
eGR metal usage	1, 2, 3
Cell density	0.7

Clock tree 2

Very Different Results

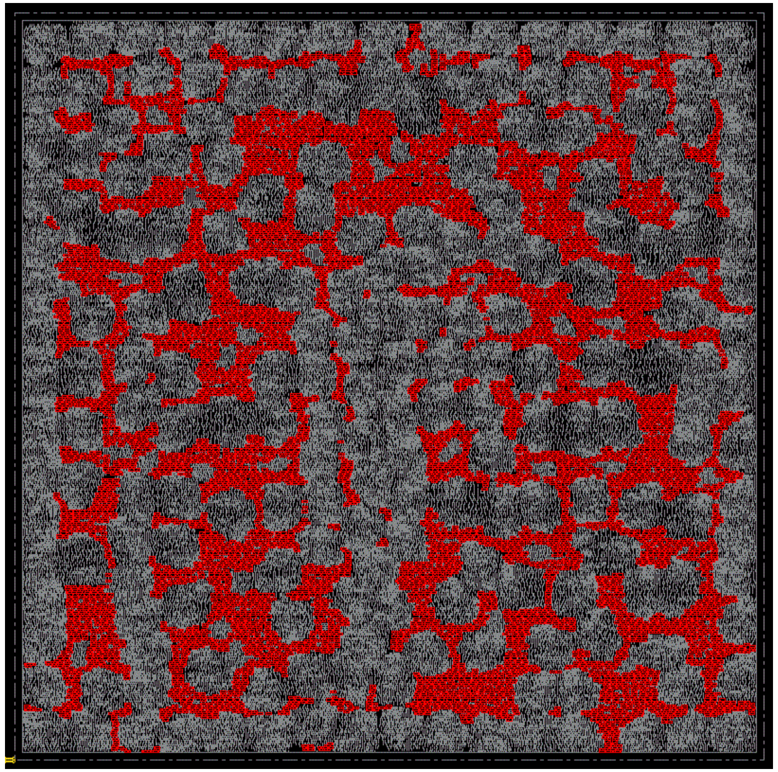


Power: 21.8mW	WL: 37.5mm
Skew: 0.15ps	Latency: 0.55ps

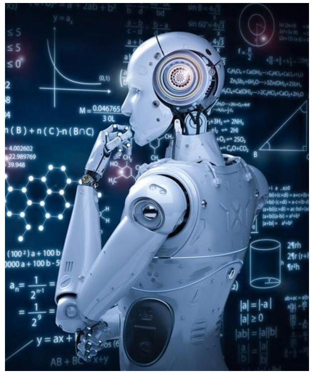


Power: 72.3mW	WL: 76.4mm
Skew: 0.13ps	Latency: 0.87ps

ML-Powered Clock Routing



placement (FF in red)



CLK WL

+

CLK skew

+

CLK power

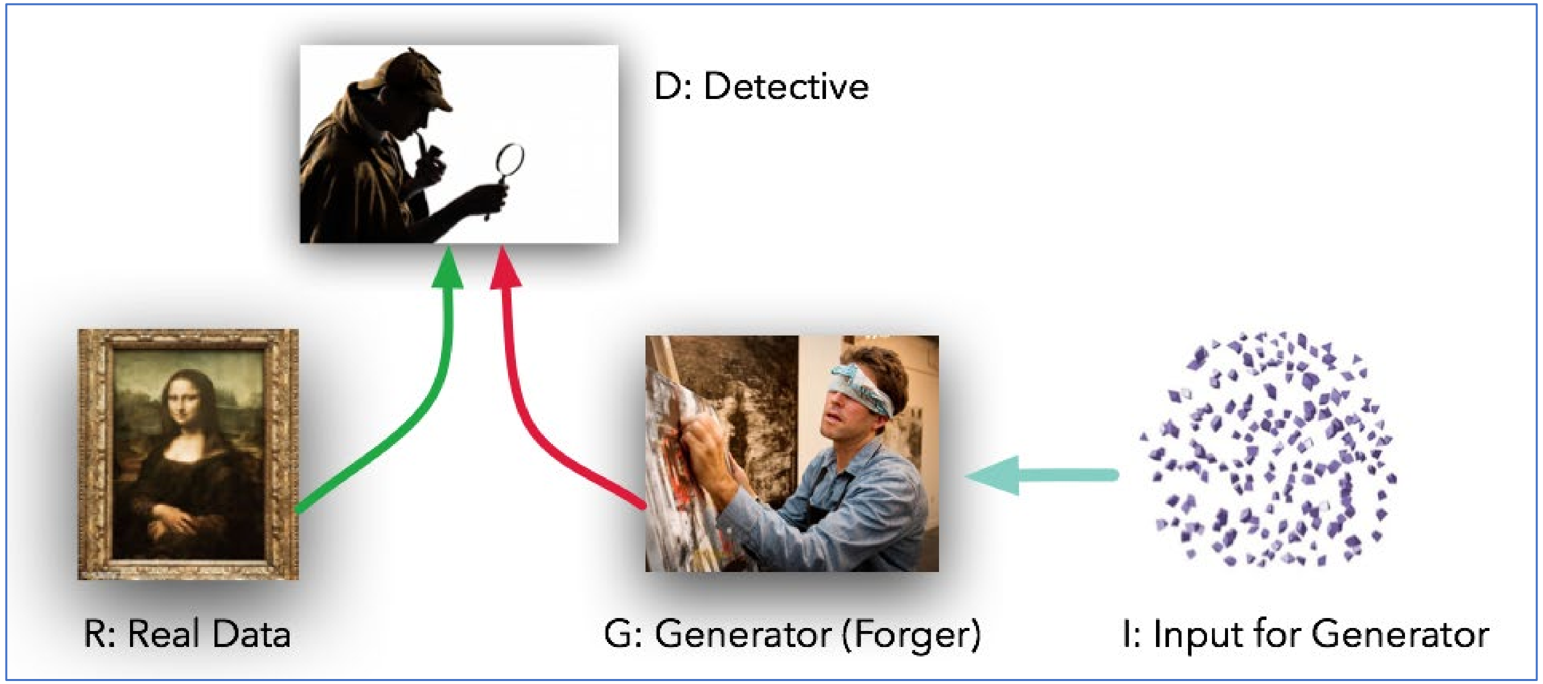
+

best setting

clock tree qualities +
best CTS parameter settings

Generative Adversarial Network (GAN)

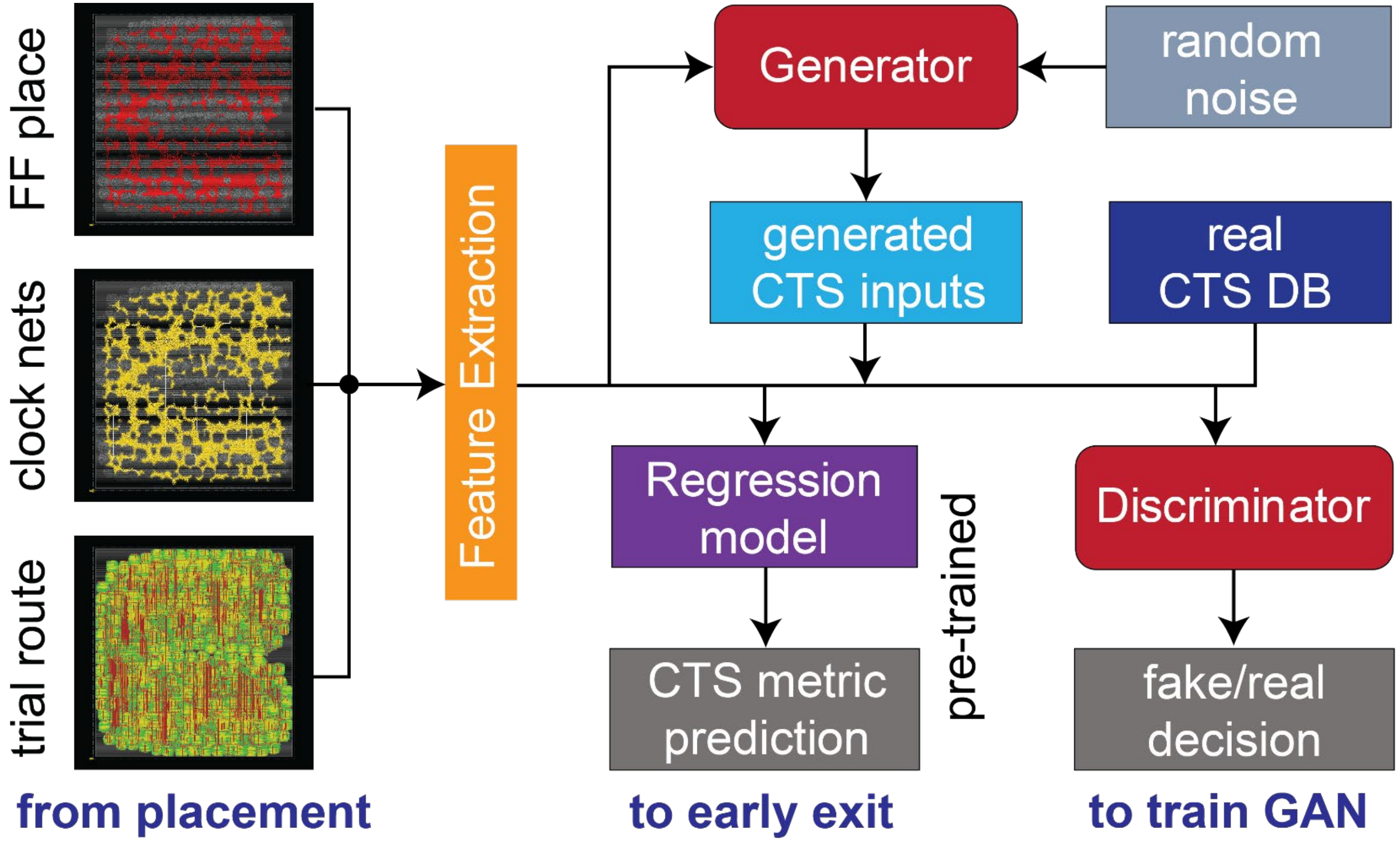
- **GAN learns to generate new data with the same statistics as the training set**
  - **Based on the "indirect" training through the discriminator**
  - **Discriminator tells how "realistic" the “fake” data is produced by the generator**
  - **Both the discriminator and generator improve through this competition**



Ian Goodfellow (2014)

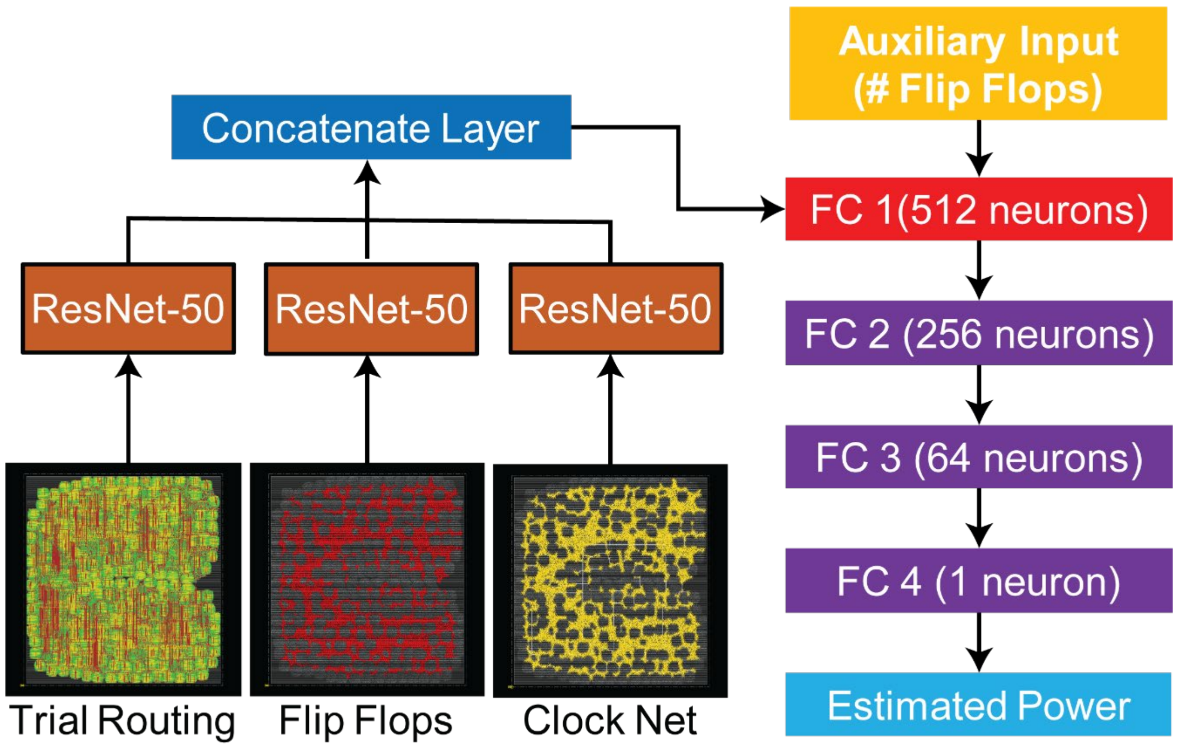
Solution: Generative Adversarial Learning

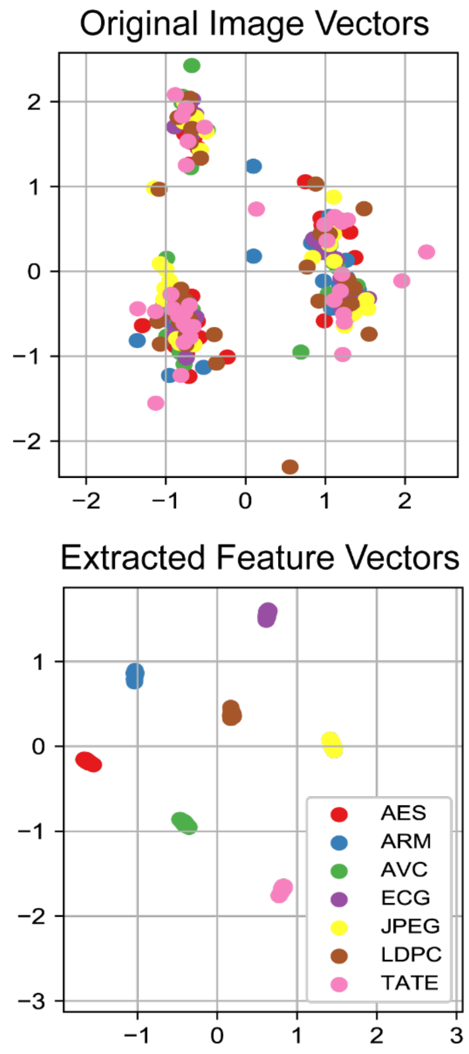
- **Conditional GAN with placement-extracted features**



Why Images?

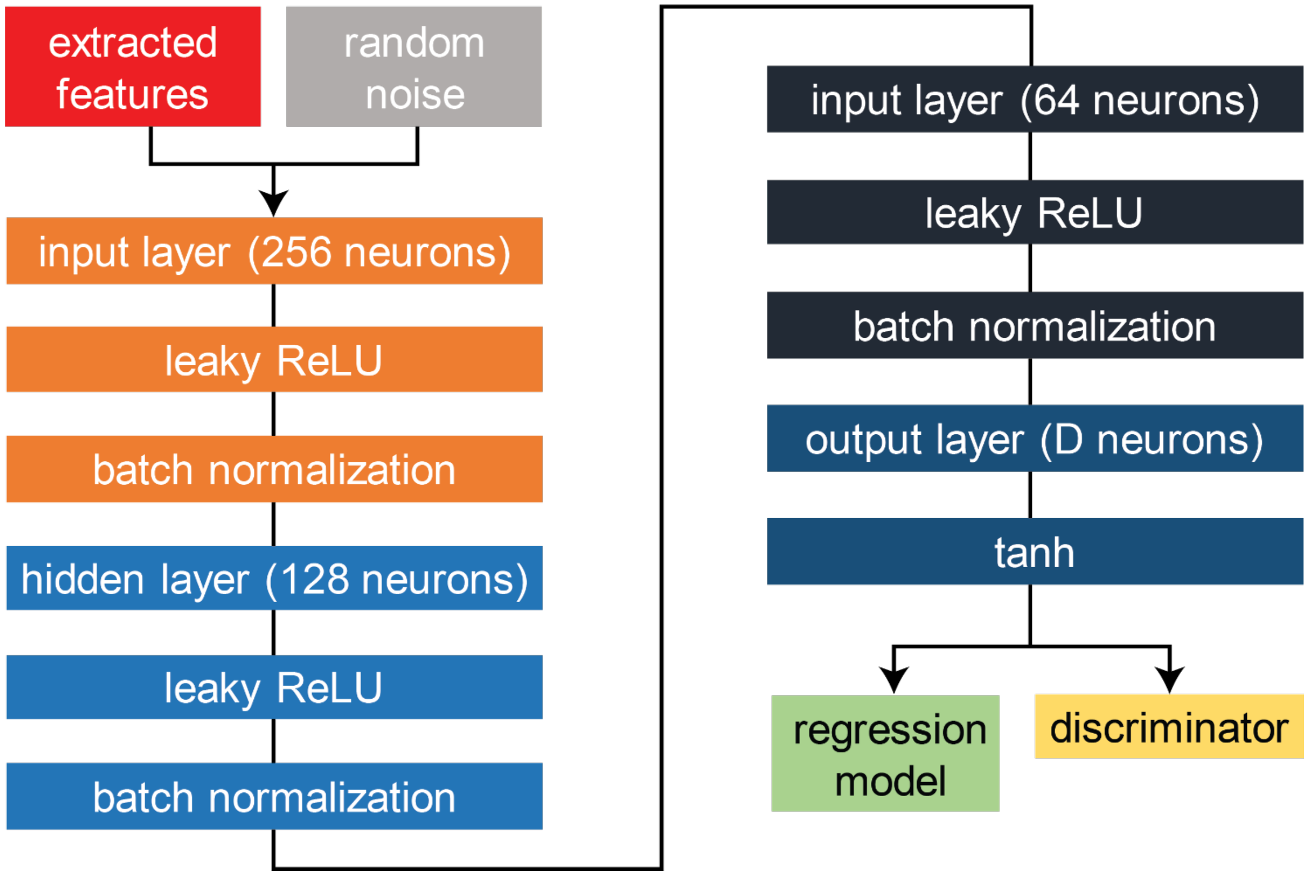
- **Useful In handling unseen netlist**
  - **Better than # FFs, # gates, # nets, etc**





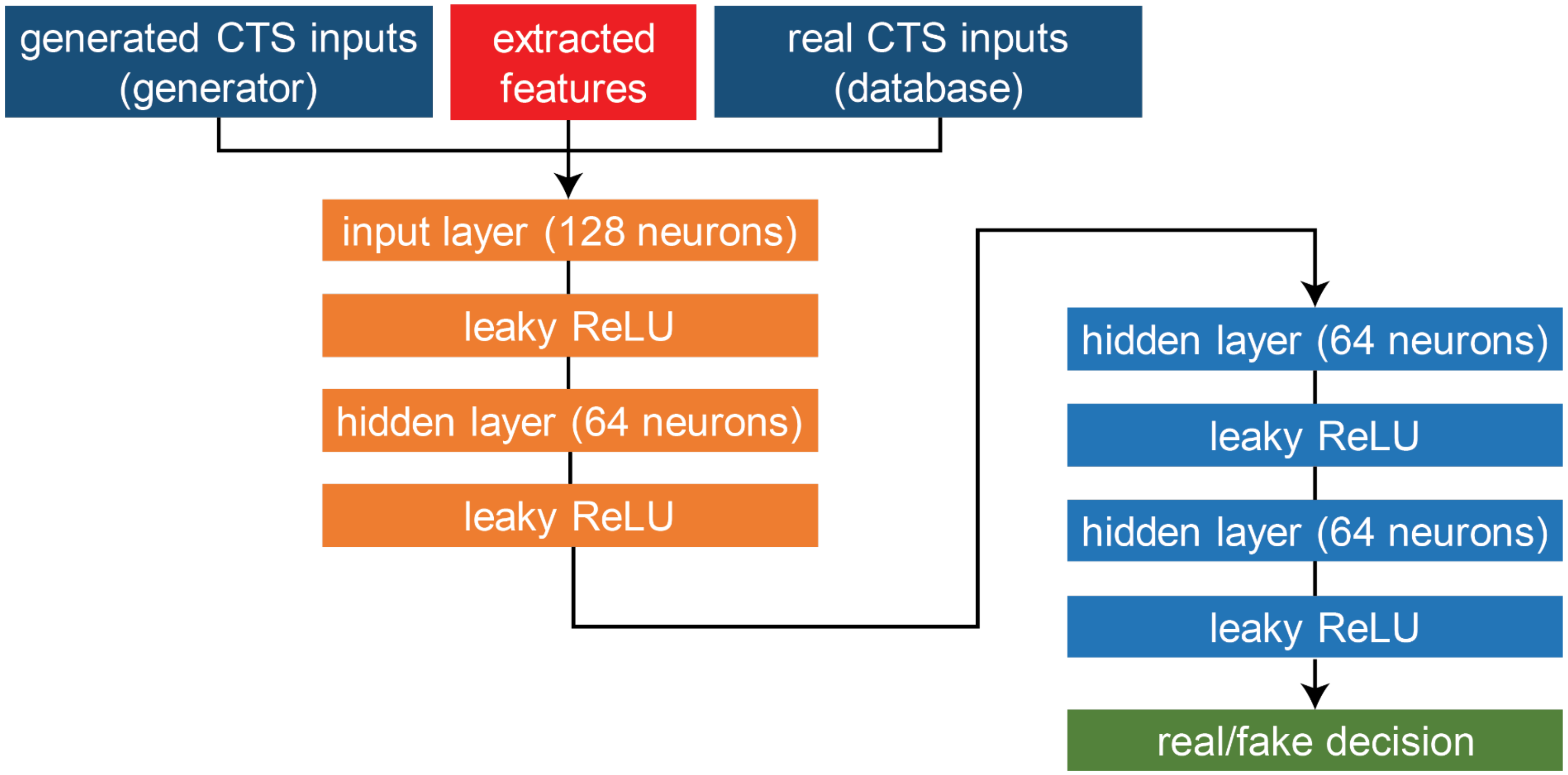
Our Generator

- **Produces “fake” CTS parameter sets**



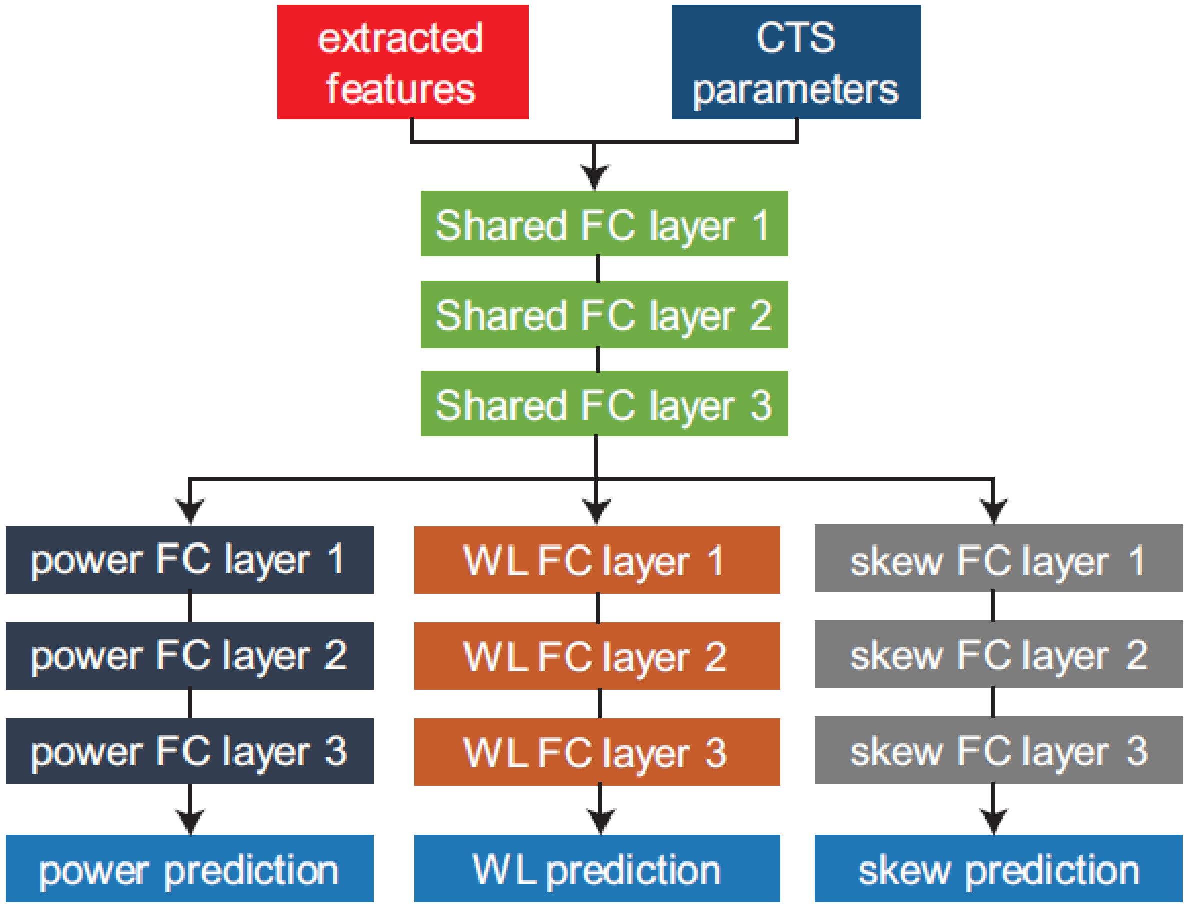
Our Discriminator

- **Catches “fake” CTS parameter sets**



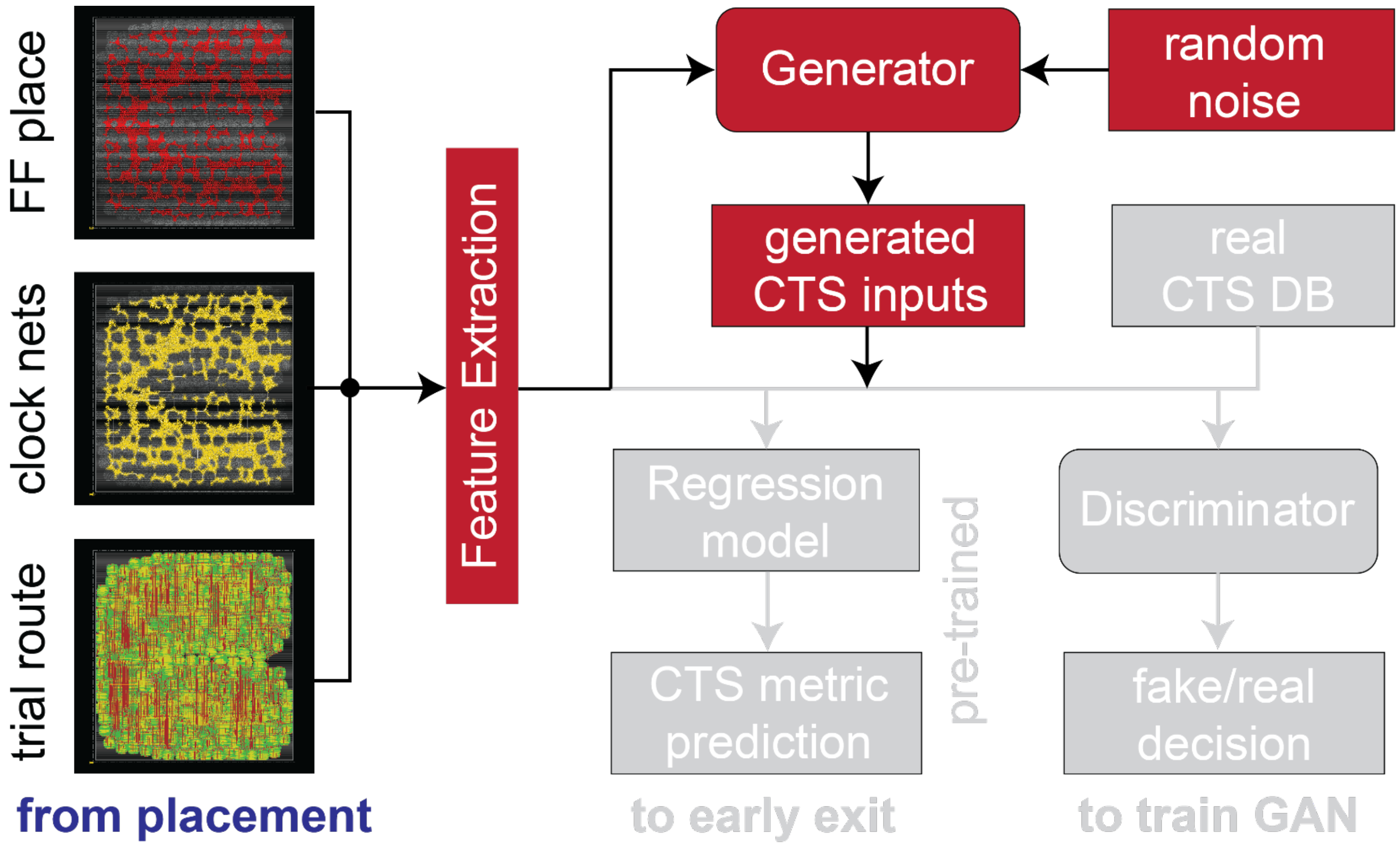
Regression Learning

- **Predicts CTS quality from CTS parameters**



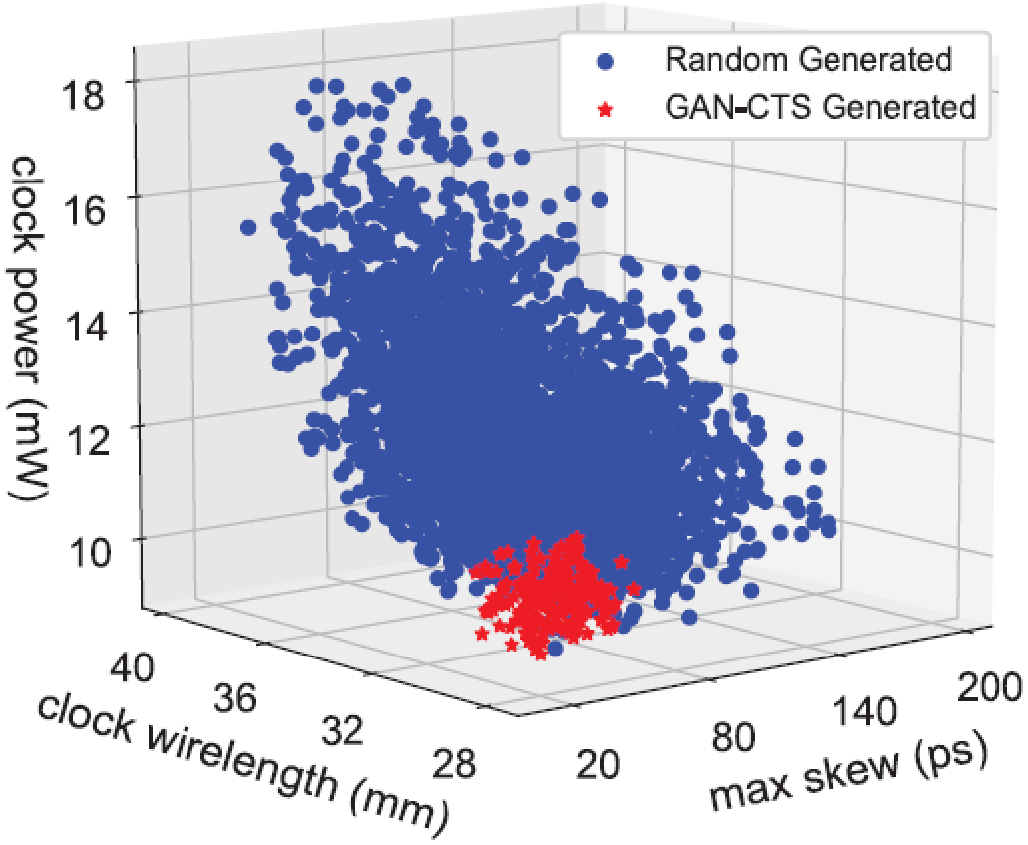
How Do We Use Trained GAN?

- **Use the generator!**

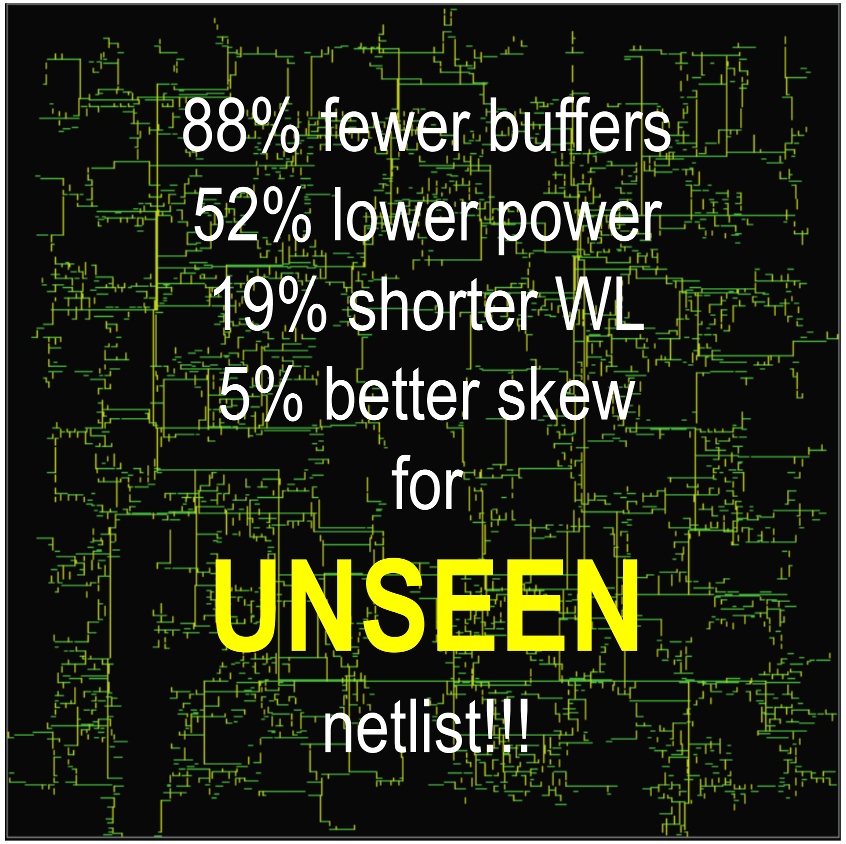


GAN-Generated Fake Clock Trees

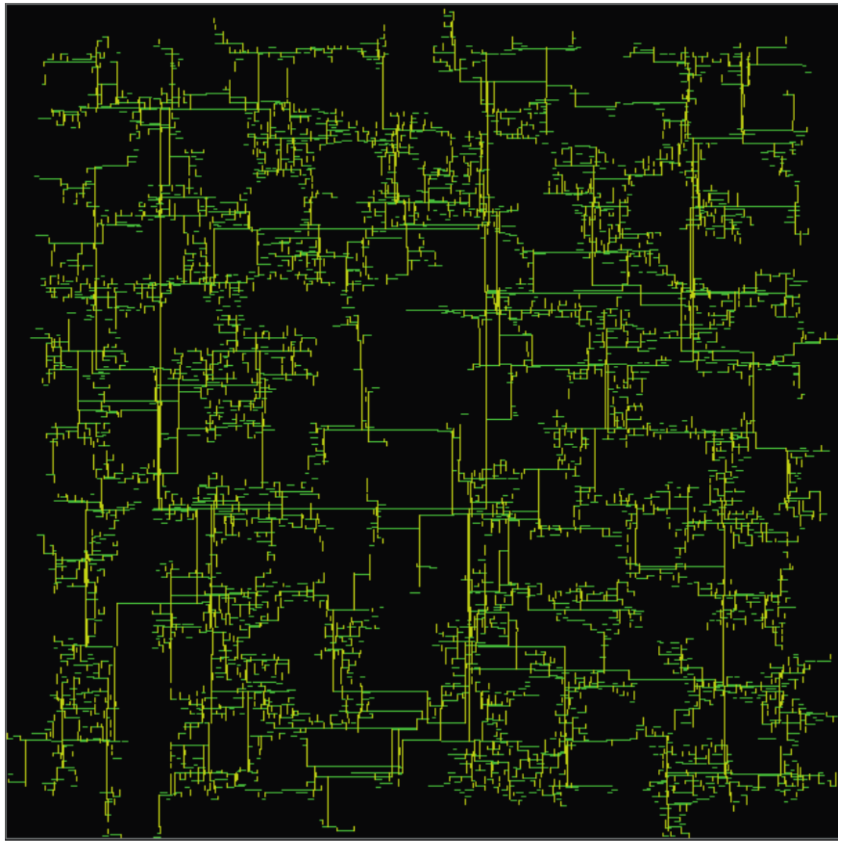
- **Fakes are of good quality**
  - In terms of power, WL, and skew
  - Useful to expand the DB!



GAN-optimized Clock Tree



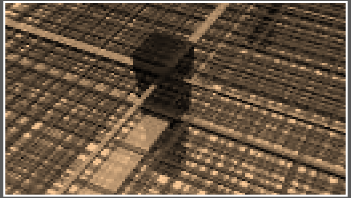
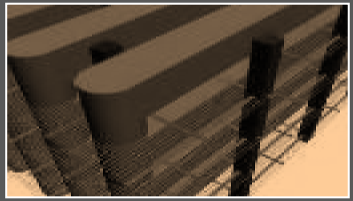
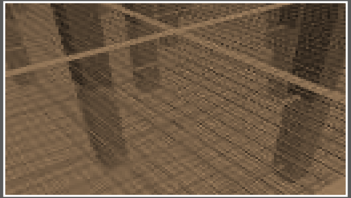
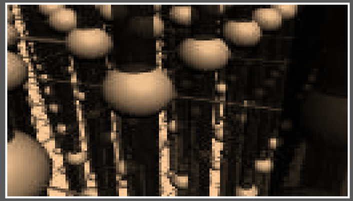
(a) GAN-CTS optimized



(b) commercial auto-setting

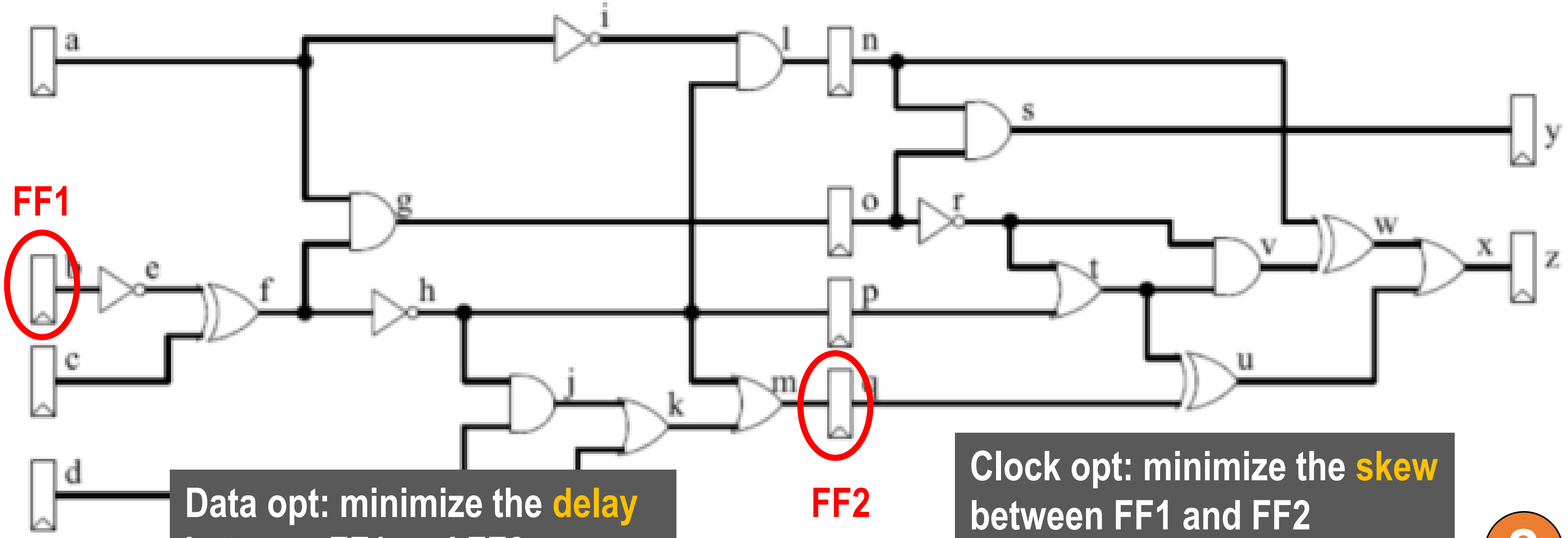
AES benchmark, TSMC 28nm, 1.1GHz clock

ML-Powered Timing Optimization

Concurrent Clock and Data Optimization

Improve both the clock and data path timing



Data opt: minimize the **delay** between FF1 and FF2

1

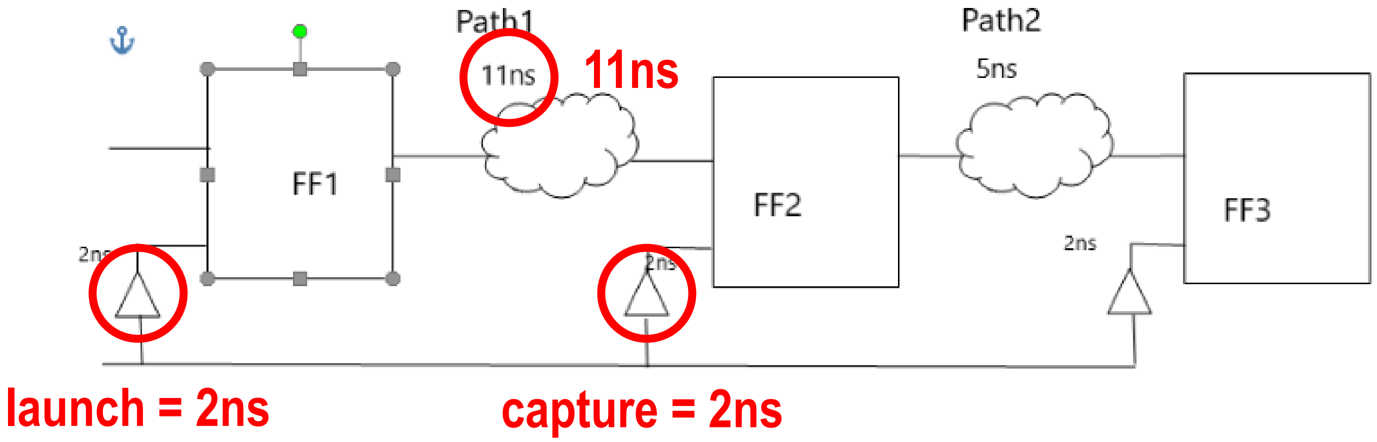
Clock opt: minimize the **skew** between FF1 and FF2

2

Useful Skew Optimization

Our target = 10ns

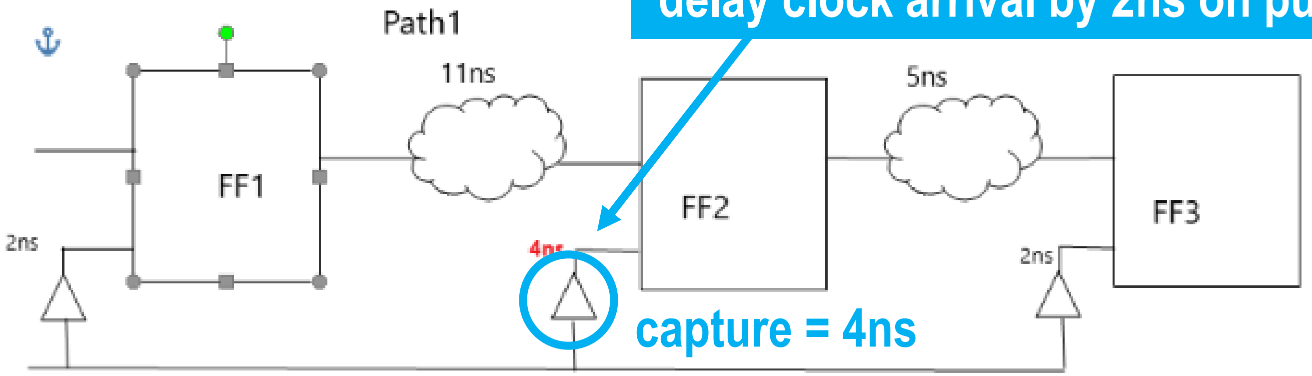
before



Required time: 2+10 = 12
Arrival time: 2+11 = 13
Skew = 12 – 13 = -1 (late)

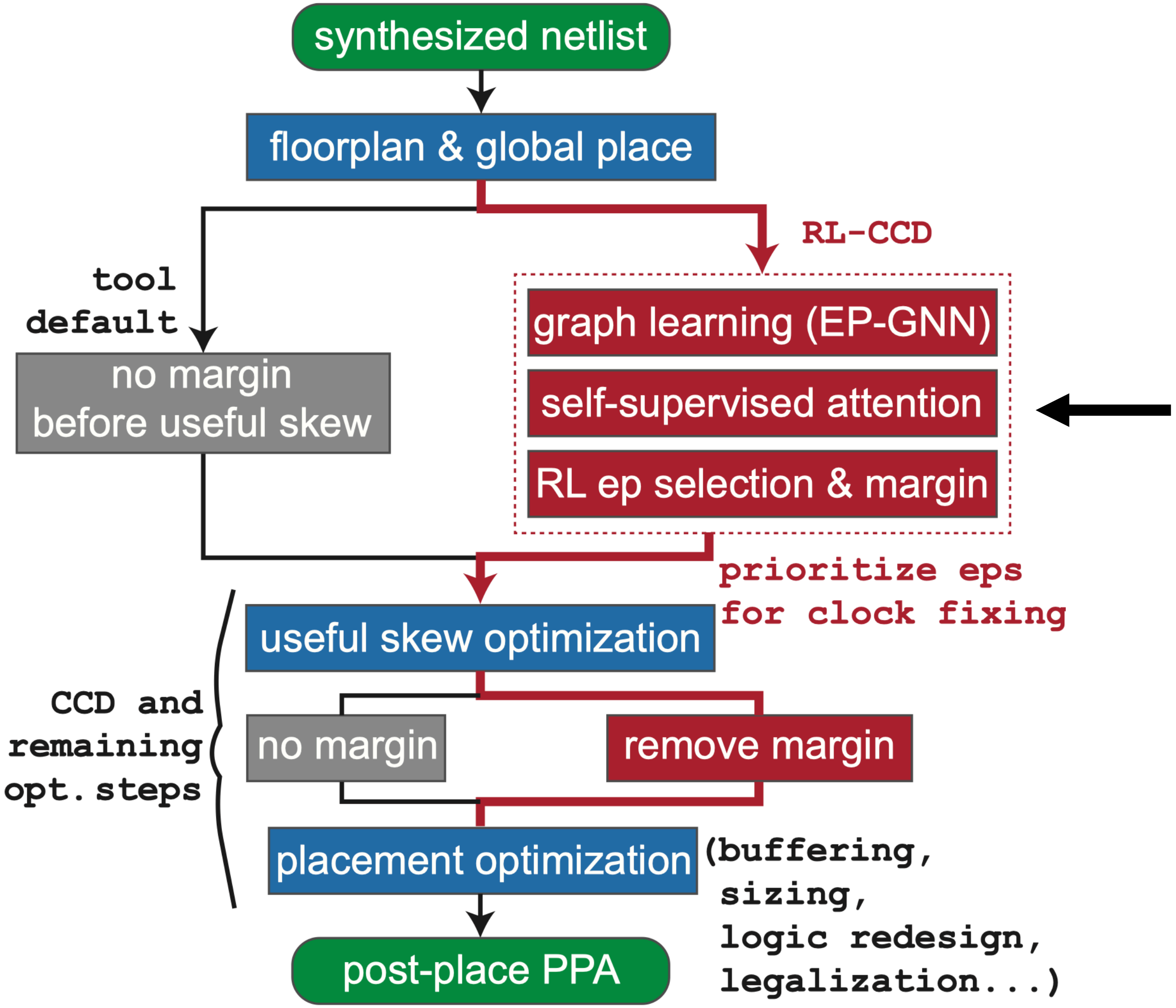
delay clock arrival by 2ns on purpose

after



Required time: 4+10 = 14
Arrival time: 2+11 = 13
Skew = 14 – 13 = 1

Our Timing Optimization Flow

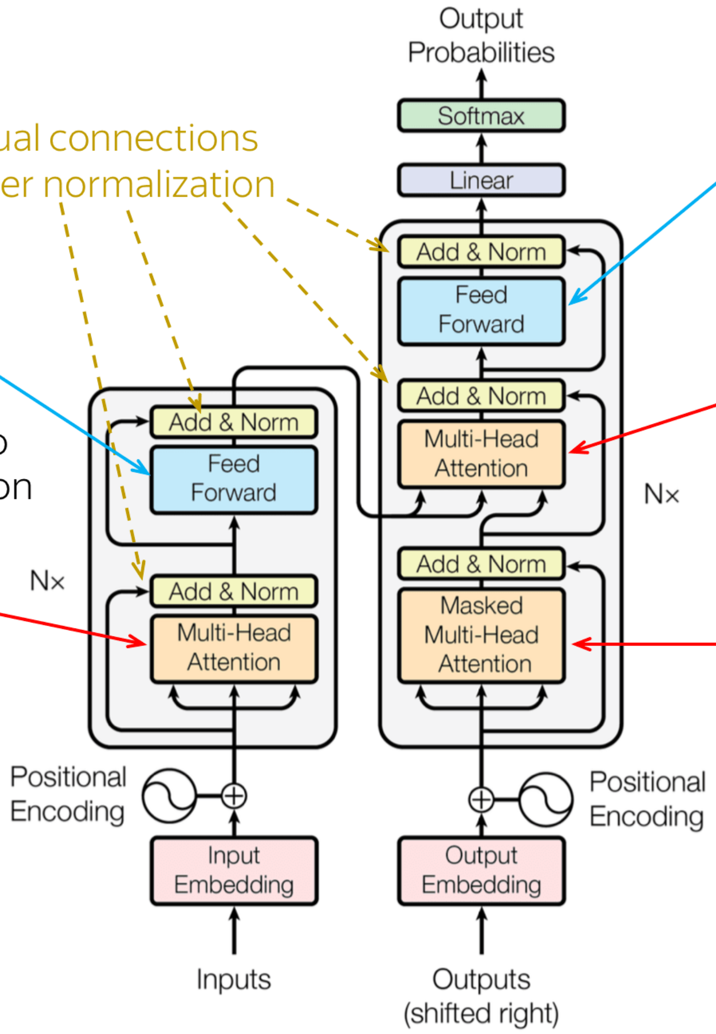


Our Idea:

- Pick a subset of end-points
- Modify their priorities on purpose
- Pass them to the subsequent skew optimizer

Transformer Architecture

Residual connections
and layer normalization



Feed-forward network:
after taking information from
other tokens, take a moment to
think and process this information

Encoder self-attention:
tokens look at each other

queries, keys, values
are computed from
encoder states

Feed-forward network:
after taking information from
other tokens, take a moment to
think and process this information

Decoder-encoder attention:
target token looks at the source

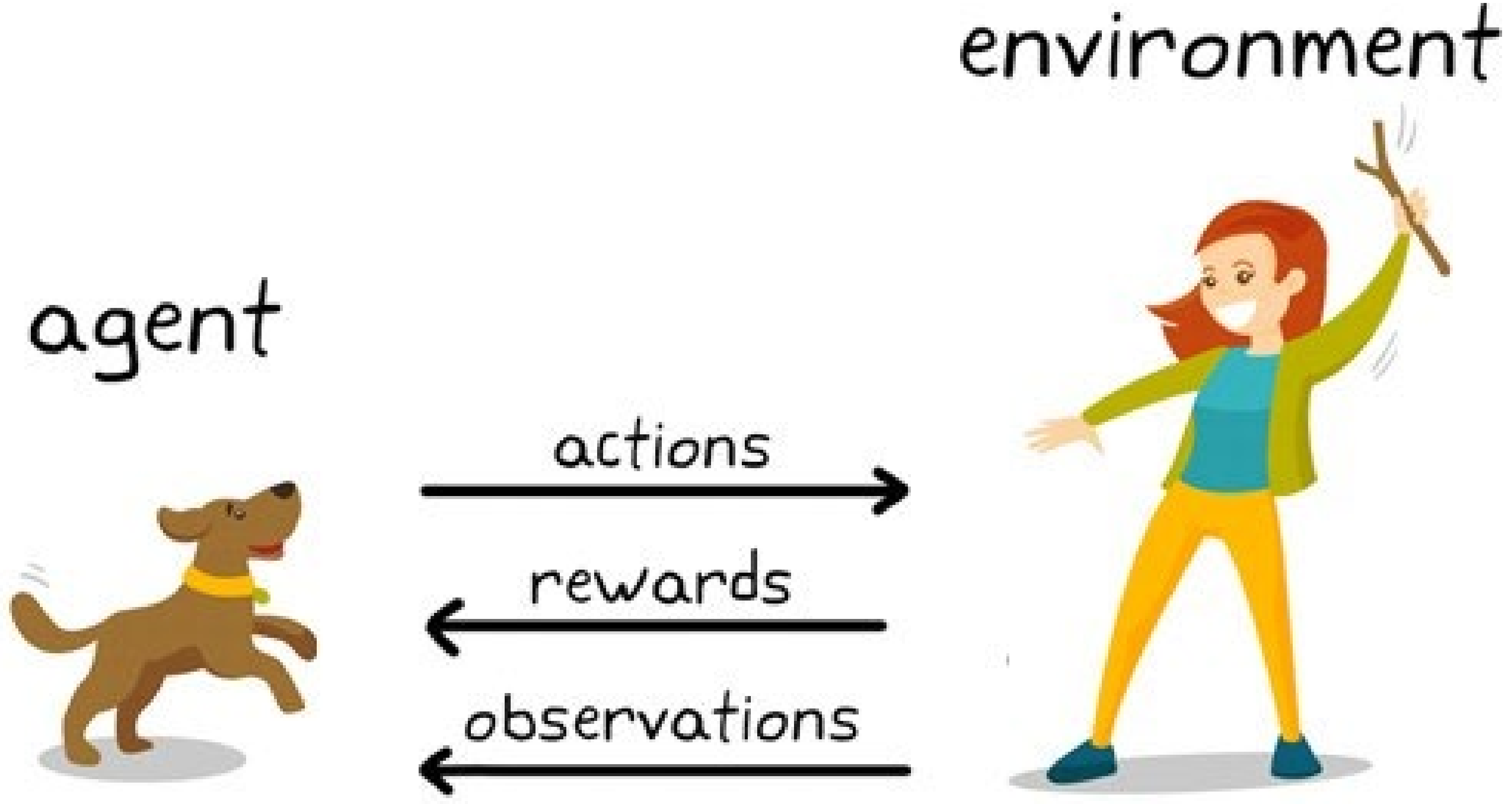
queries – from decoder states; keys
and values from encoder states

Decoder self-attention (masked):
tokens look at the previous tokens

queries, keys, values are computed
from decoder states

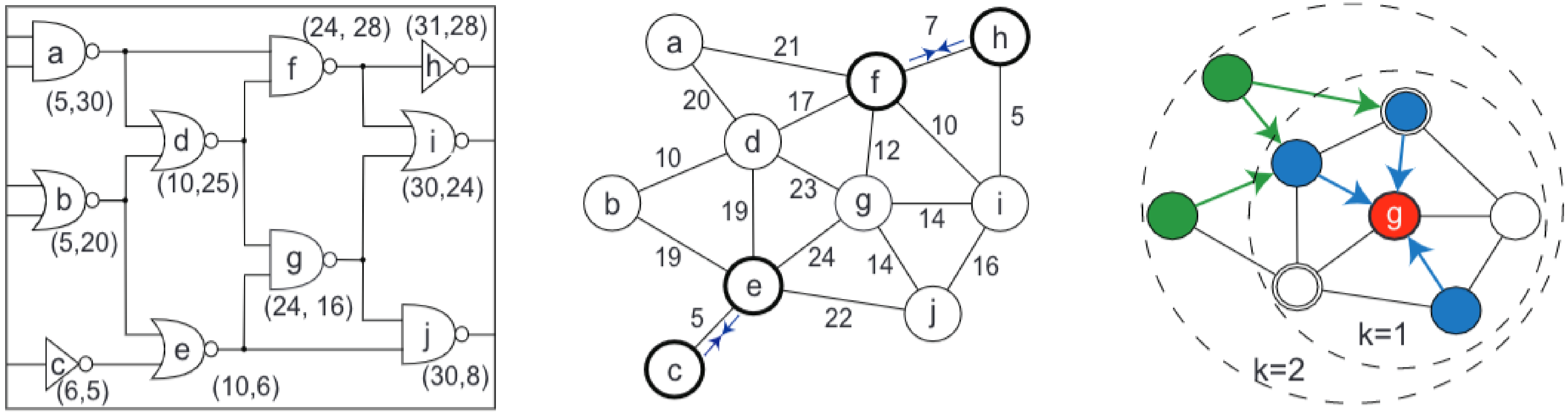
Reinforcement Learning (RL)

- **One of the 3 main approaches in machine learning**
  - **Key benefit: no data needed to learn from!**
  - **Key drawback: slow..**



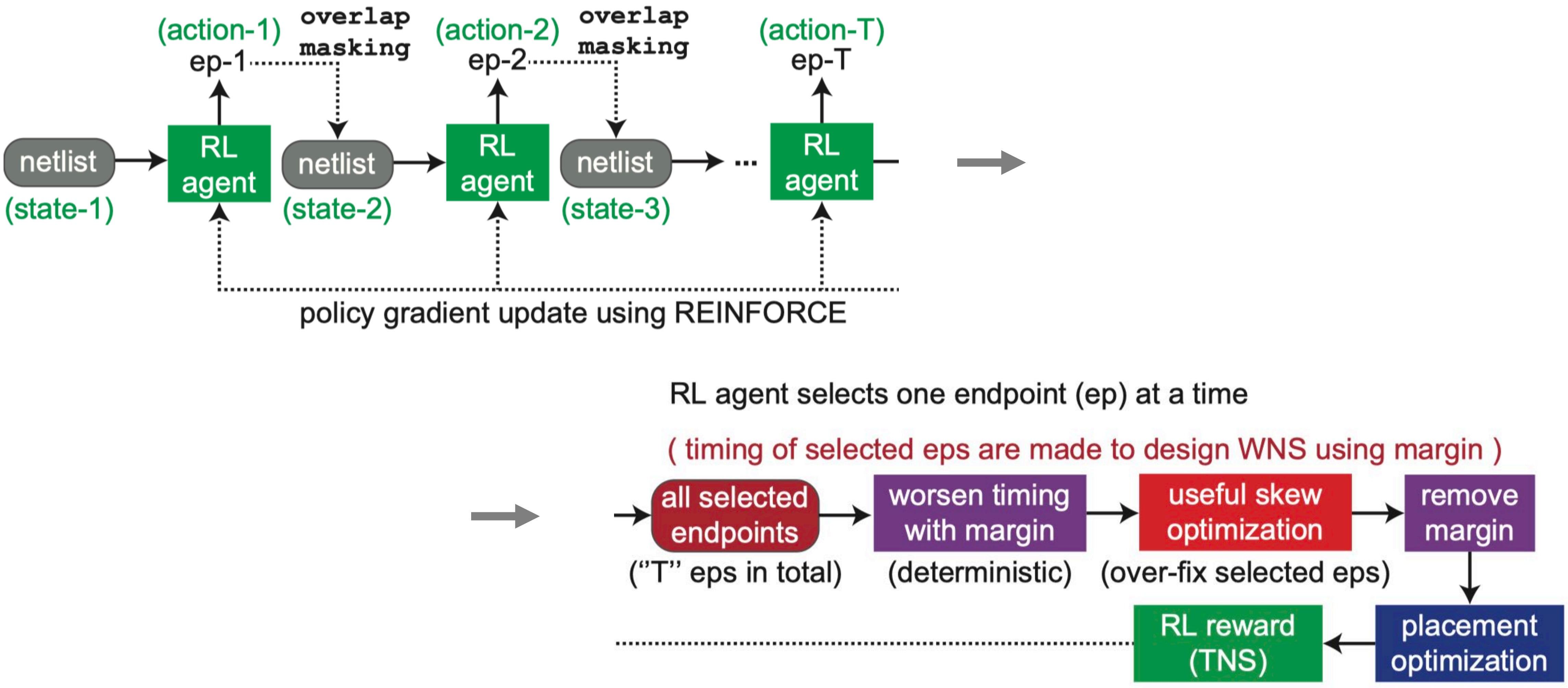
Graph Neural Network (GNN)

- **Learn from neighbors**
  - **Digital circuits are graphs, naturally**
  - **SO, very popular in circuit design community**

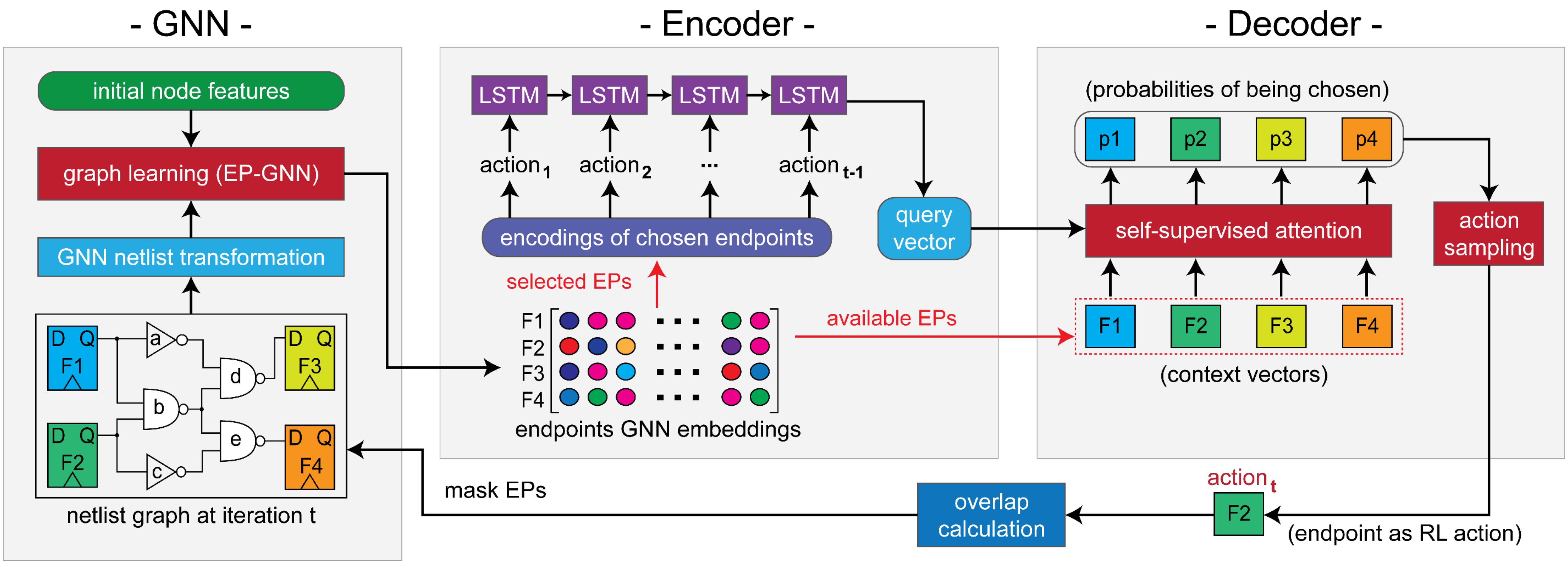


(a) (b) (c)

RL Strategy to Pick Endpoints



Our Transformer Architecture

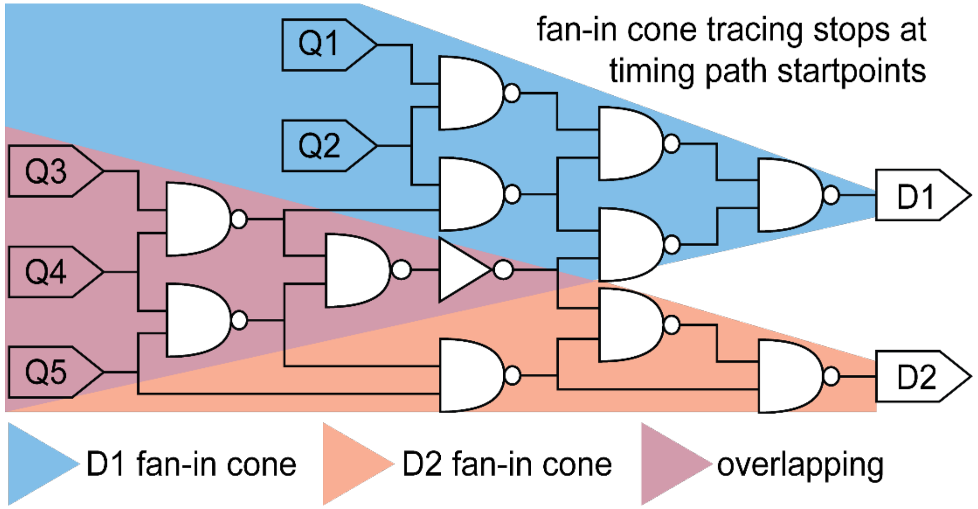


* Loop continues until all endpoints are either selected or masked

Some Details

name	# dim.	description
RL masked	1	is selected or masked by RL-CCD
locations	2	cell (x,y) location in global placement
outNet cap	1	output net capacitance
load cap	1	sum of driving load capacitance
cell cap	1	cell input capacitance
cell power	2	cell internal power and leakage power
net power	1	output net switching power
max toggle	1	maximum toggle rate at output pin
wst slack	1	worst slack of paths through cell
wst output slew	1	worst output transition
wst input slew	1	worst input transition

Initial node features to be further optimized in our GNN



We avoid selecting endpoints that share too many common gates.

Experimental Results

design (# cells)	begin (post global place)				default tool flow (16 threads)					RL-CCD enhanced (ours)				
	WNS	TNS	#vio. EPs	total power	WNS	TNS (goal)	#vio. EPs	total power	run-time	WNS	TNS (goal)	#vio. EPs	total power	run-time
block1 (577K)	-0.24	-2009.98	33785	482.92	-0.16	-97.2	4296	1114.33	1.00	-0.16	-84.0 (-14.1%)	3603	1116.48	16
block2 (1.3M)	-0.18	-1104.03	40091	761.41	-0.05	-2.93	540	764.13	1.00	-0.07	-2.56 (-12.6%)	443	763.98	36
block3 (353K)	-0.26	-2966.04	36265	468.06	-0.17	-149.28	4119	474.72	1.00	-0.18	-87.45 (-41.42%)	1942	473.80	29
block4 (370K)	-0.46	-4590.85	38943	297.19	-0.11	-20.78	1258	322.48	1.00	-0.12	-7.40 (-64.4%)	421	321.97	31
block5 (194K)	-0.27	-1165.33	9708	199.45	-0.14	-162.45	4271	205.50	1.00	-0.14	-59.99 (-63.1%)	2081	204.95	39
block6 (195K)	-0.30	-1382.51	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible].03%)	1146	119.50	20
block7 (416K)	-0.34	-2108.89	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]3.6%)	1009	134.35	21
block8 (135K)	-0.15	-1186.14	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]5.0%)	2314	349.56	42
block9 (162K)	-0.11	-50.90	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]0.7%)	44	114.55	8
block10 (84K)	-0.43	-4428.41	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]7.6%)	3603	90.69	45
block11 (180K)	-0.29	-793.53	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]8.8%)	135	276.79	32
block12 (243K)	-0.32	-1720.92	18465	78.72	-0.19	-102.90	2223	27.83	1.00	-0.18	-79.9 (-22.4%)	1794	27.83	46
block13 (507K)	-0.12	-375.08	12987	63.48	-0.06	-39.37	3779	64.95	1.00	-0.06	-33.72 (-14.4%)	3291	64.80	10
block14 (816K)	-0.16	-1913.75	44044	333.60	-0.06	-51.43	4260	340.07	1.00	-0.06	-48.89 (-4.9%)	3915	340.00	7
block15 (821K)	-0.18	-331.51	11002	66.17	-0.11	-40.55	2116	66.72	1.00	-0.11	-37.78 (-6.83%)	1861	66.71	20
block16 (432K)	-0.18	-374.15	9228	27.18	-0.07	-32.24	2586	28.09	1.00	-0.05	-24.89 (-22.8%)	2149	28.09	16
block17 (507K)	-0.14	-226.09	8860	407.69	-0.07	-46.22	2472	412.26	1.00	-0.06	-33.05 (-28.5%)	2361	412.21	35
block18 (412K)	-0.41	-2787.22	51675	583.88	-0.10	-6.14	123	1183.46	1.00	-0.10	-5.81 (-5.4%)	124	1182.23	26
block19 (922K)	-0.16	-383.69	8009	98.66	-0.09	-19.01	667	218.38	1.00	-0.06	-13.71 (-27.9%)	626	218.33	47
											avg. -24%	avg. -19%	avg. -0.2%	

24% TNS improvement on average (64% max) on 19 commercial designs implemented using 5 – 12nm

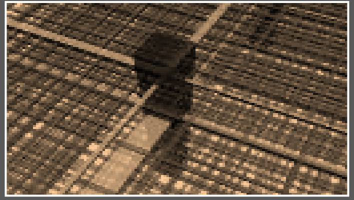
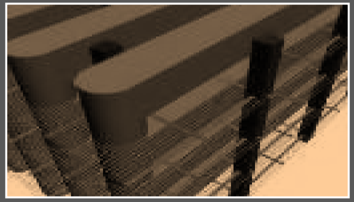
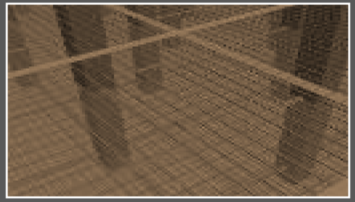
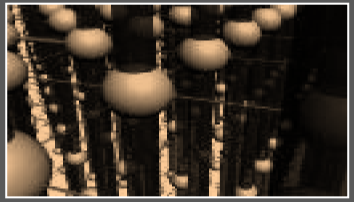
Conclusions

- **ML-Powered VLSI Clock Routing: GAN**
  - Image-based feature extraction
  - Outperformed commercial auto-setting

- **ML-Powered Timing Optimization: RL + Transformer**
  - AI-based end-point selection
  - Significant improvement on 19 commercial designs in 5nm to 12nm

- **ML-Powered VLSI Circuit Placement: RL + Attention**
  - Attention-based knob tuning
  - Outperformed multi-arm bandit & human expert

ML-Powered Circuit Placement

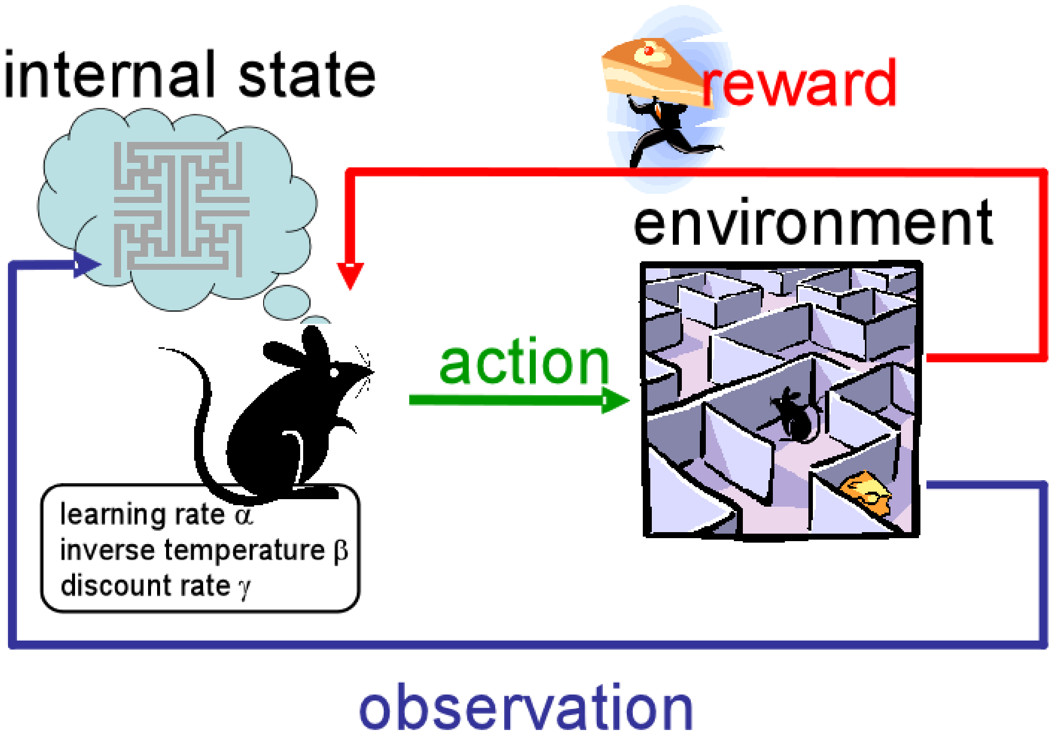
Placement Parameters

- **12 placement parameters from Cadence Innovus**
  - **6 billions combinations**

Name	Objective	Type	Groups	# val
eco max distance	maximum distance allowed during placement legalization	integer	detail	[0, 100]
legalization gap	minimum sites gap between instances	integer	detail	[0, 100]
max density	controls the maximum density of local bins	integer	global	[0, 100]
eco priority	instance priority for refine place	enum	detail	3
activity power driven	level of effort for activity power driven placer	enum	detail + effort	3
wire length opt	optimizes wirelength by swapping cells	enum	detail + effort	3
blockage channel	creates placement blockages in narrow channels between macros	enum	global	3
timing effort	level of effort for timing driven placer	enum	global + effort	2
clock power driven	level of effort for clock power driven placer	enum	global + effort	3
congestion effort	the effort level for relieving congestion	enum	global + effort	3
clock gate aware	specifies that placement is aware of clock gate cells in the design	bool	global	2
uniform density	enables even cell distribution	bool	global	2

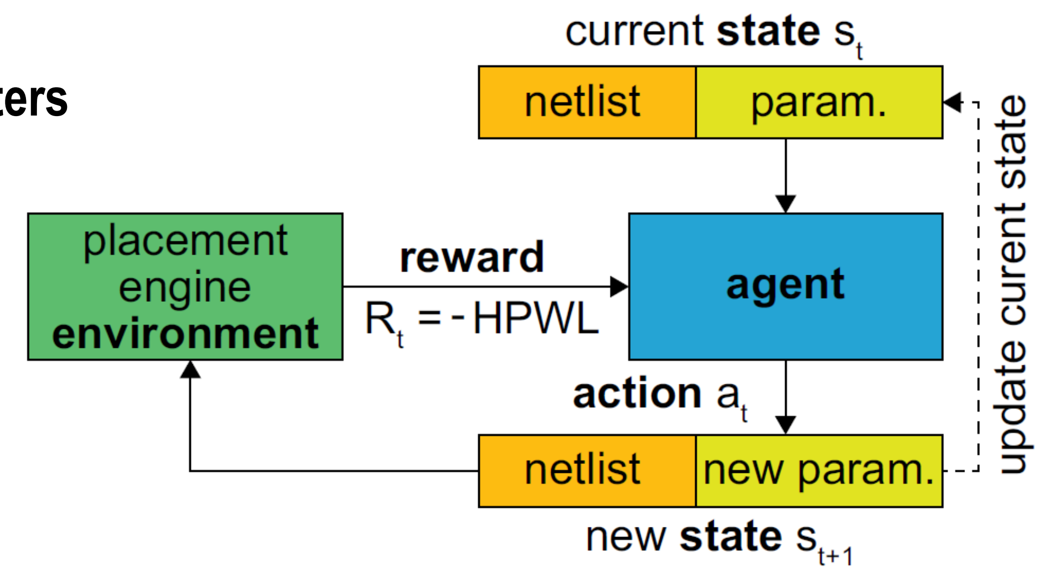
Reinforcement Learning

- RL agent learns in an interactive environment
  - By trial and error
  - Using feedback from its own actions and experiences



Our RL Framework

- Goal: minimize half-perimeter wirelength (HPWL) after placement
- States
  - Set of all netlists and all possible placement parameter settings
- Actions
  - Set of actions that modifies the current parameters
- State transition
  - The next state is the same netlist with updated parameters
- Reward
  - HPWL improvement



Our Actions and Reward

11 actions

1. FLIP Booleans
2. UP Integers
3. DOWN Integers
4. UP Efforts
5. DOWN Efforts
6. UP Detailed
7. DOWN Detailed
8. UP Global (does not touch the bool)
9. DOWN Global (does not touch the bool)
10. INVERT-MIX timing vs. congestion vs. WL efforts
11. DO NOTHING

single action changes multiple parameters

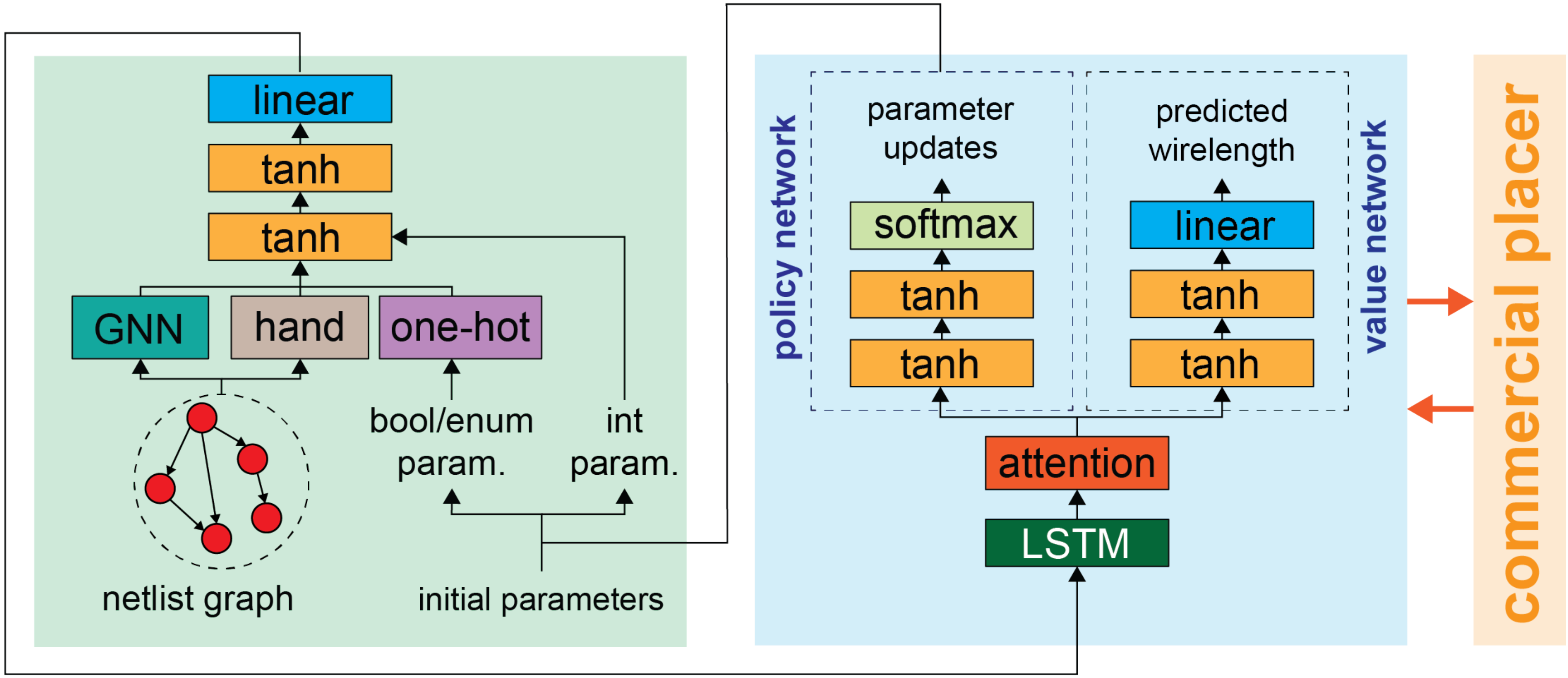
$$R_t := \frac{HPWL_{\text{Human Baseline}} - HPWL_t}{HPWL_{\text{Human Baseline}}}$$

reward function (saving over human design)

Name	Objective	Type	Groups	# val
eco max distance	maximum distance allowed during placement legalization	integer	detail	[0, 100]
legalization gap	minimum sites gap between instances	integer	detail	[0, 100]
max density	controls the maximum density of local bins	integer	global	[0, 100]
eco priority	instance priority for refine place	enum	detail	3
activity power driven	level of effort for activity power driven placer	enum	detail + effort	3
wire length opt	optimizes wirelength by swapping cells	enum	detail + effort	3
blockage channel	creates placement blockages in narrow channels between macros	enum	global	3
timing effort	level of effort for timing driven placer	enum	global + effort	2
clock power driven	level of effort for clock power driven placer	enum	global + effort	3
congestion effort	the effort level for relieving congestion	enum	global + effort	3
clock gate aware	specifies that placement is aware of clock gate cells in the design	bool	global	2
uniform density	enables even cell distribution	bool	global	2

Our Agent Architecture

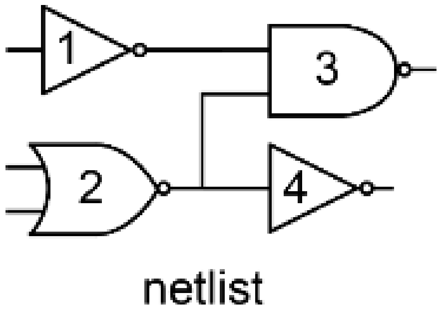
- **LSTM-based neural network**
  - **Policy network: updates placement parameters**
  - **Value network: predicts WL**

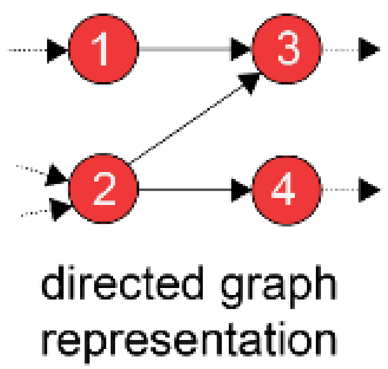


20 Graph-Related Features

- **Netlist metadata**
  - Metadata from netlist
- **Topological features**
  - Extracted from netlist graph (directed)

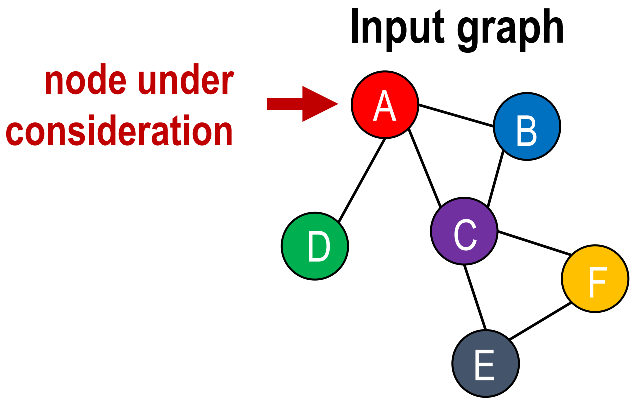
Metadata (10)		Topological (10)	
Name	Type	Name	Type
# cells	integer	average degree	float
# nets	integer	average fanout	float
# cell pins	integer	largest SCC	integer
# IO	integer	max. clique	integer
# nets w. fanout $\in]5, 10[$	integer	chromatic nb.	integer
# nets w. fanout ≥ 10	integer	max. logic level	integer
# FFs	integer	RCC	float
total cell area (um^2)	integer	$\overline{CC}$	float
# hardmacros	integer	Fiedler value	float
macro area (um^2)	integer	spectral radius	float





32 GNN Features Using GraphSAGE

- **Our initial features (5):**
  - gate type, degree, fanout, area, delay



$$\mathbf{h}_v^0 = \mathbf{x}_v$$

initial layer 0 embeddings are equal to node features

$$\mathbf{h}_v^k = \sigma\left(\mathbf{W}_k \sum_{u \in N(v)} \frac{\mathbf{h}_u^{k-1}}{|N(v)|} + \mathbf{B}_k \mathbf{h}_v^{k-1}\right)$$

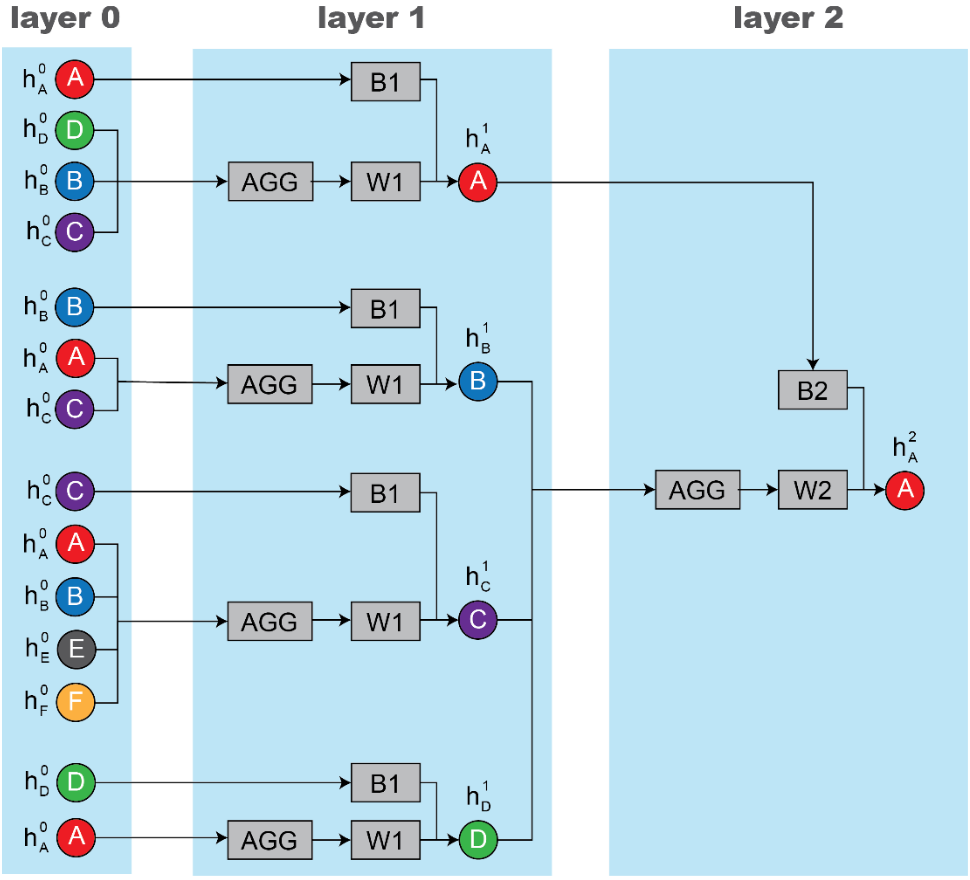
k^{th} layer embedding of v

non-linearity

average of neighbor's previous layer embeddings

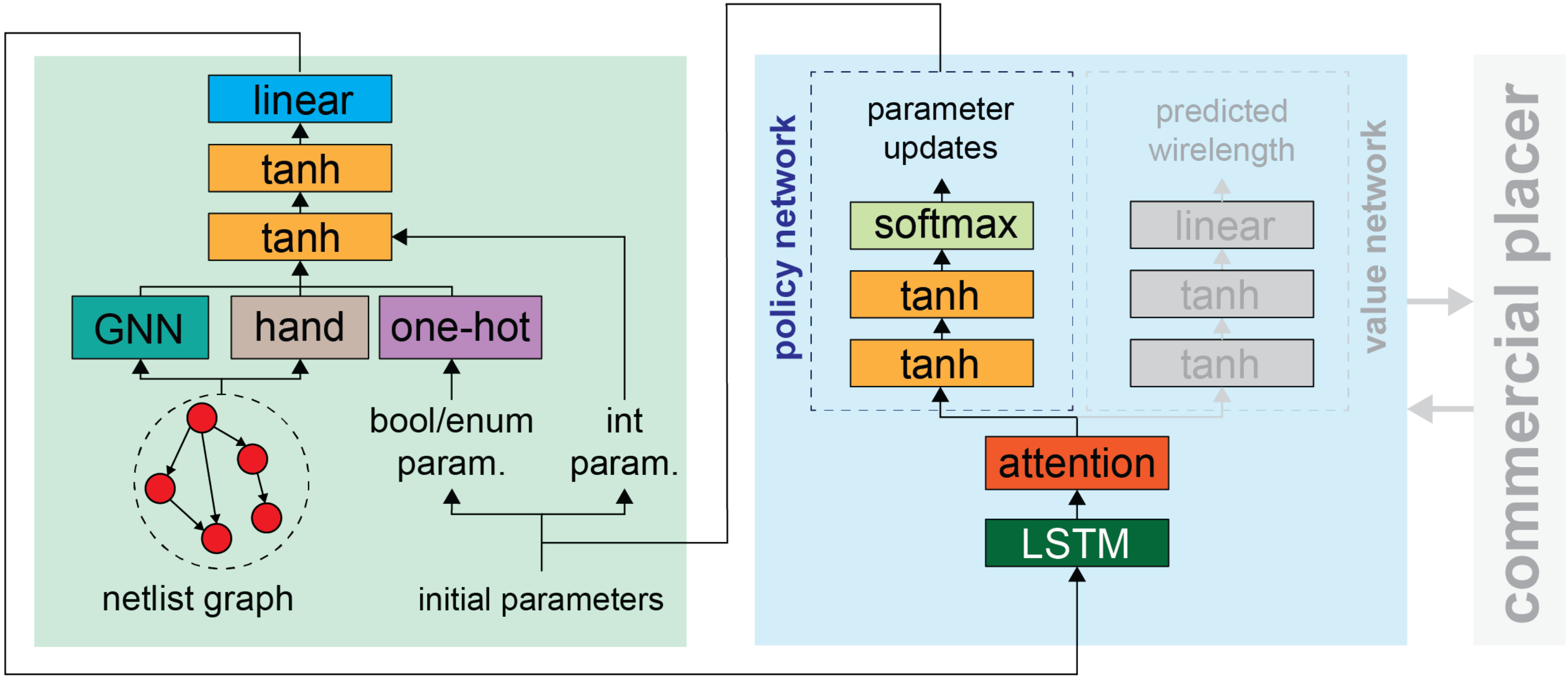
previous layer embedding of v

$$z_v = \mathbf{h}_v^{last}$$



How Do We Use Trained RL Agent?

- Use our policy network!
  - Iteratively improve a random set using trained RL
  - We stop if DO NOTHING is issued 3 times in a row



Experimental Setting

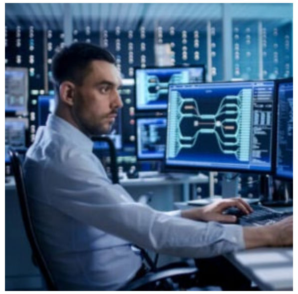
- **TSMC 28nm**
- **Diversity in netlists**
  - **13 without macros**
  - **2 with macros**
  - **Macros are pre-placed manually**

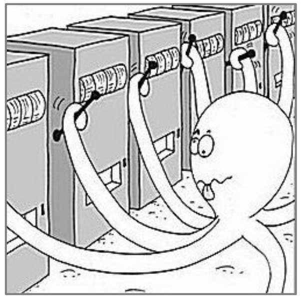
Name	#cells	#nets	#IO	RCC_3	LL	Sp. R.	RT
training set							
PCI	1.2K	1.4K	361	510	17	25.6	0.5
DMA	10K	11K	959	65	25	26.4	1
B19	33K	34K	47	19	86	36.1	2
DES	47K	48K	370	14	16	25.6	2
VGA	52K	52K	184	15	25	26.5	3
ECG	83K	84K	1.7K	7.5	23	26.8	4
Rocket	92K	95K	377	8.1	42	514.0	6
AES	112K	112K	390	5.8	14	102.0	6
Nova	153K	155K	174	4.6	57	11,298	9
Tate	187K	188K	1.9K	3.2	21	25.9	10
JPEG	239K	267K	67	2.8	30	287.0	12
test set (unseen netlist)							
LDPC	39K	41K	4.1K	18	19	328.0	2
OpenPiton	188K	196K	1.6K	3.9	76	3940	19
Netcard	300K	301K	1.8K	2.9	32	27.3	24
Leon3	326K	327K	333	2.4	44	29.5	26

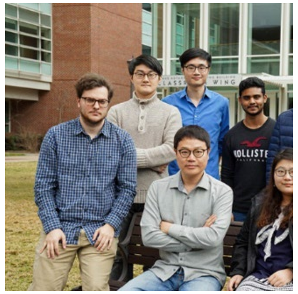
Training Results

- **Training time**
  - ~100 hours doing 14,400 placements with Innovus (16 parallel runs)
  - MAB: Nelder-Mead, Differential Evolution, Simulated Annealing, Genetic Algorithm, Particle Swarm

Netlist	human	MAB [1]	(Δ%)	RL	(Δ%)
PCI	0.010	0.0092	(−8.0%)	0.0092	(−8.0%)
DMA	0.149	0.139	(−6.7%)	0.135	(−9.4%)
B19	0.30	0.28	(−6.7%)	0.28	(−6.7%)
DES	0.42	0.37	(−11.9%)	0.36	(−14.3%)
VGA	1.52	1.40	(−7.9%)	1.41	(−7.2%)
ECG	0.72	0.65	(−9.7%)	0.68	(−5.5%)
Rocket	1.33	1.27	(−4.5%)	1.20	(−9.8%)
AES	1.49	1.44	(−2.7%)	1.40	(−6.0%)
AVC-Nova	1.59	1.49	(−6.3%)	1.46	(−8.2%)
Tate	1.53	1.42	(−7.2%)	1.45	(−5.2%)
JPEG	2.14	1.96	(−8.4%)	1.88	(−12.2%)

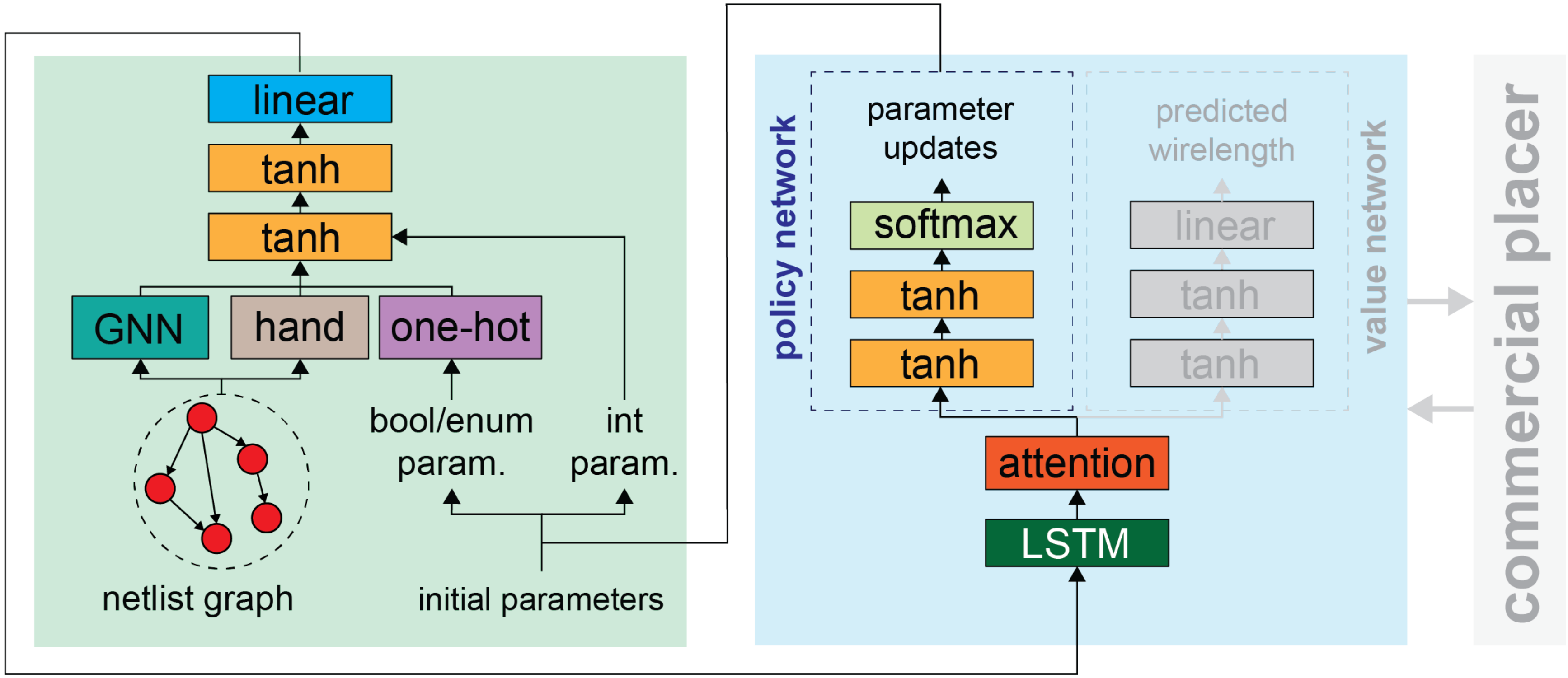




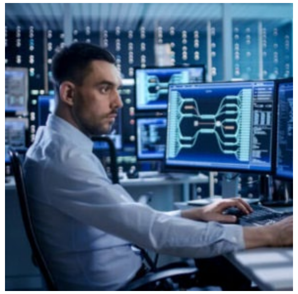


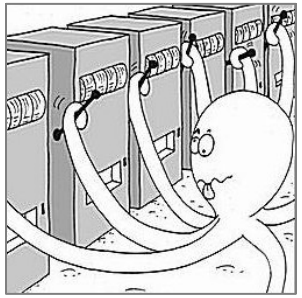
How Do We Use Trained RL Agent?

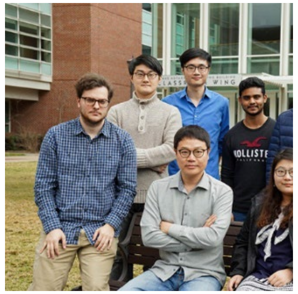
- Use our policy network!
  - Iteratively improve a random set using trained RL
  - We stop if DO NOTHING is issued 3 times in a row

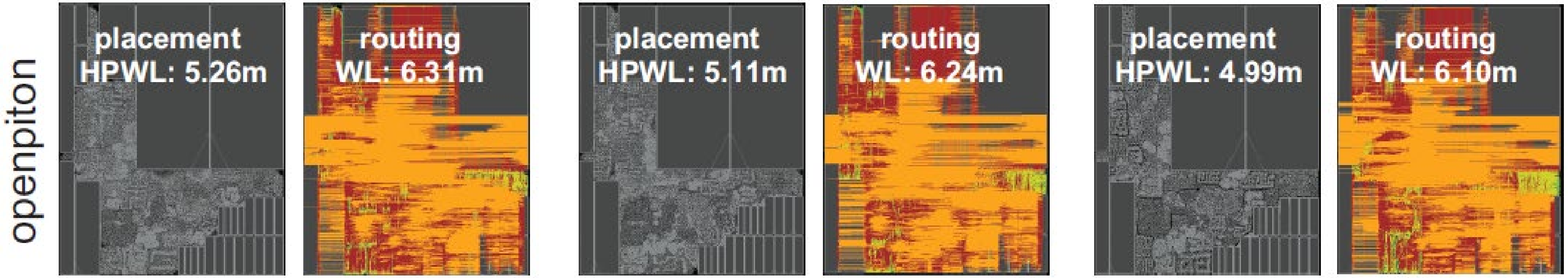


What About After Routing?

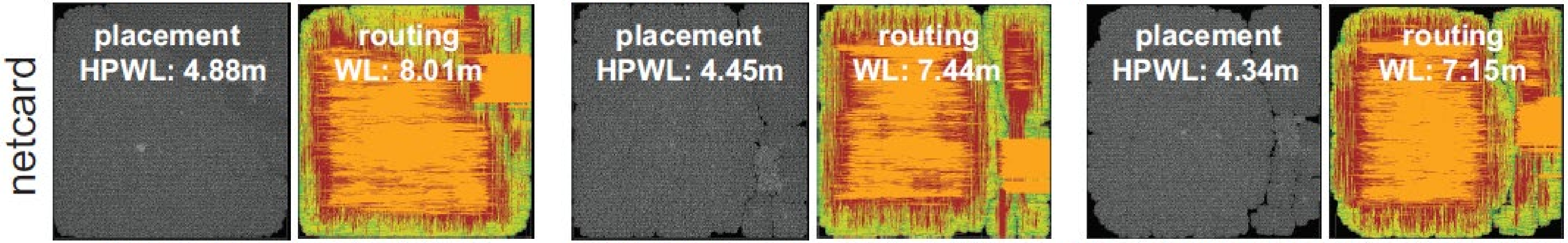








(a) human design (took 7hrs) (b) Multi-Armed Bandit (took 16hrs) (c) reinforcement learning (took 20min)



(a) human design (took 8hrs) (b) Multi-Armed Bandit (took 20hrs) (c) reinforcement learning (took 25min)