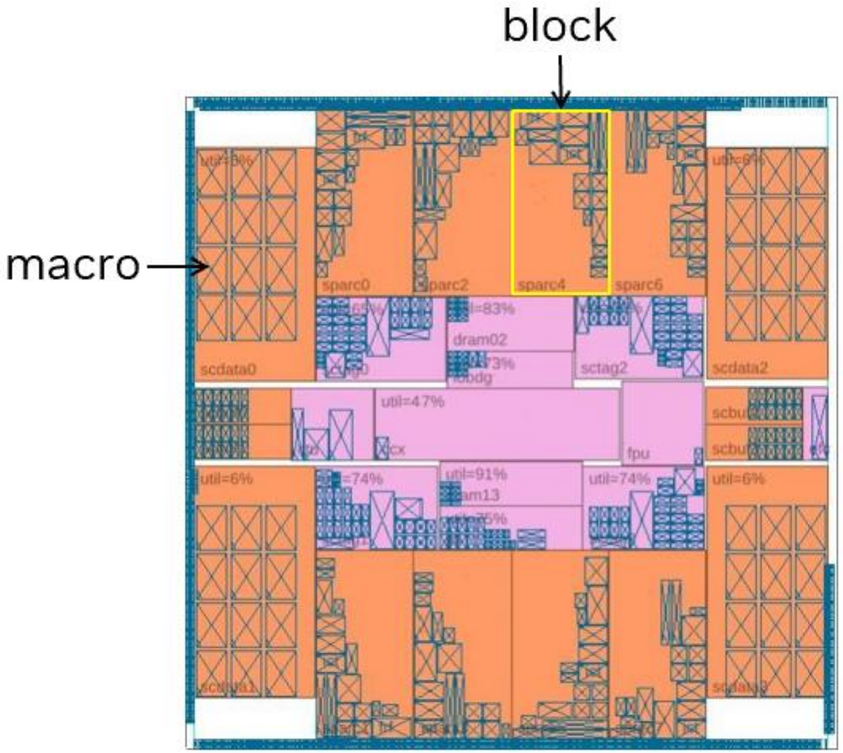
Scaling Physical Design with Generative AI and Gradient-based Methods on GPU

Haoxing (Mark) Ren

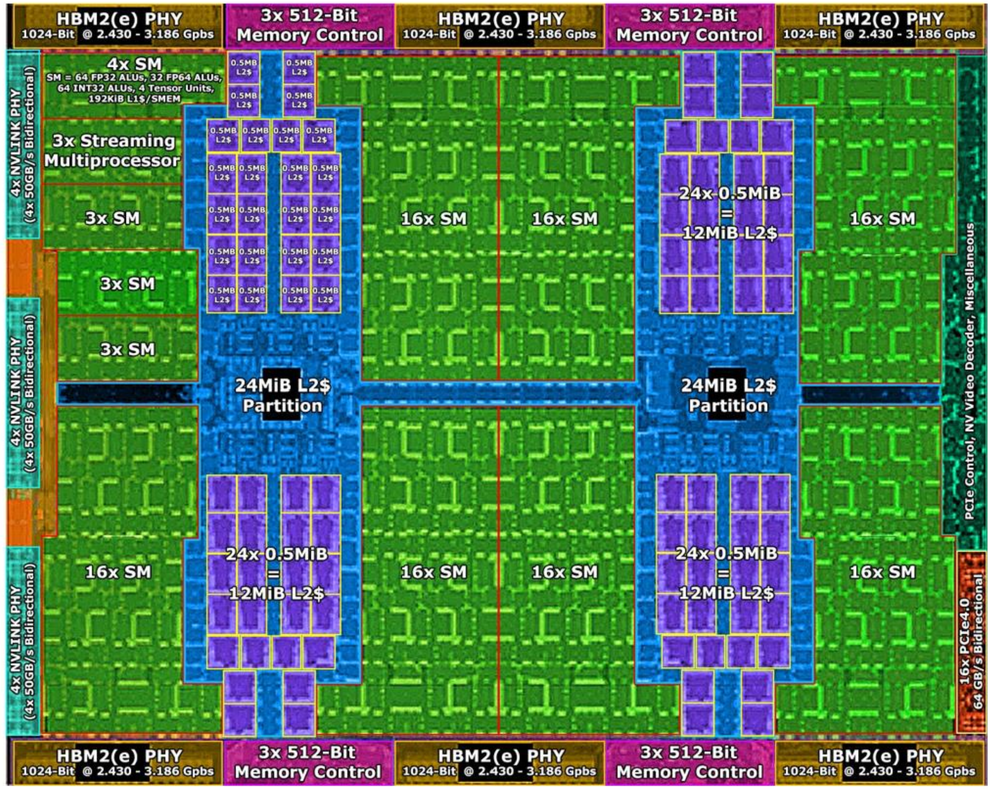
Physical Design at Block Level

Divide-and-conquer: physical implementation with design blocks



Ultra Sparc T1

- 279M Transistors
- 20+ blocks
- 10+ macros/block

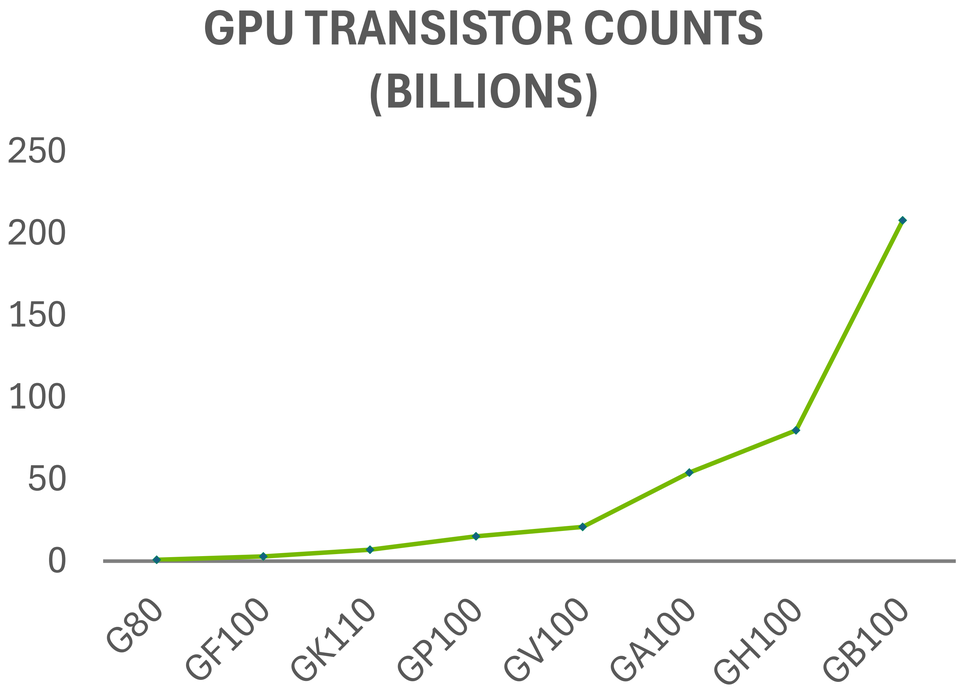


A100

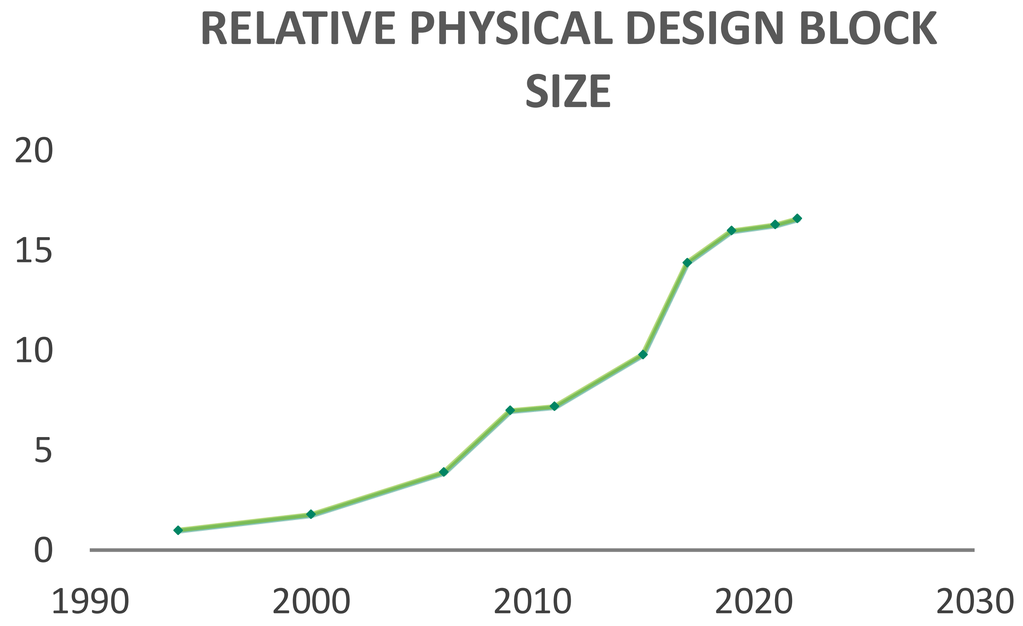
- 54B Transistors
- 100s blocks
- 100s macros/block

Physical Design Scalability Challenge

Mainly because of P&R tools scalability



Design size continue to scale



Average physical design block size levels off
>1 week turn around time for 1-2 Million cell block

Sameer Halepete, Accelerating the Design and Performance of Next Generation Computing Systems with GPUs, ISPD'22

Accelerate Placement on GPU

Nonlinear Placement Algorithm / Gradient-based Method

Minimize wirelength and density overflow

$$\min \quad f = WL(x,y) + \mu \sum_{bin} Overflow_{bin}(x,y)^2$$

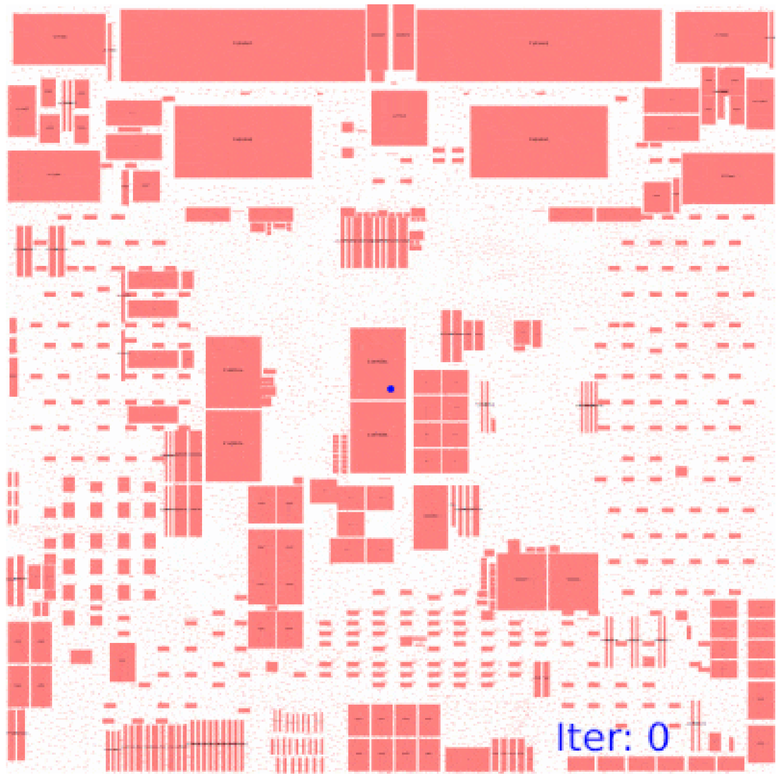
Gradient-based optimization :

$$x = x - \gamma \frac{\partial}{\partial x} f$$

$$y = y - \gamma \frac{\partial}{\partial y} f$$

Neural networks back propagation

$$\min \quad \sum_{i} loss(w, x_i)^2 + L_2(w)$$



Accelerate with DL Frameworks on GPU

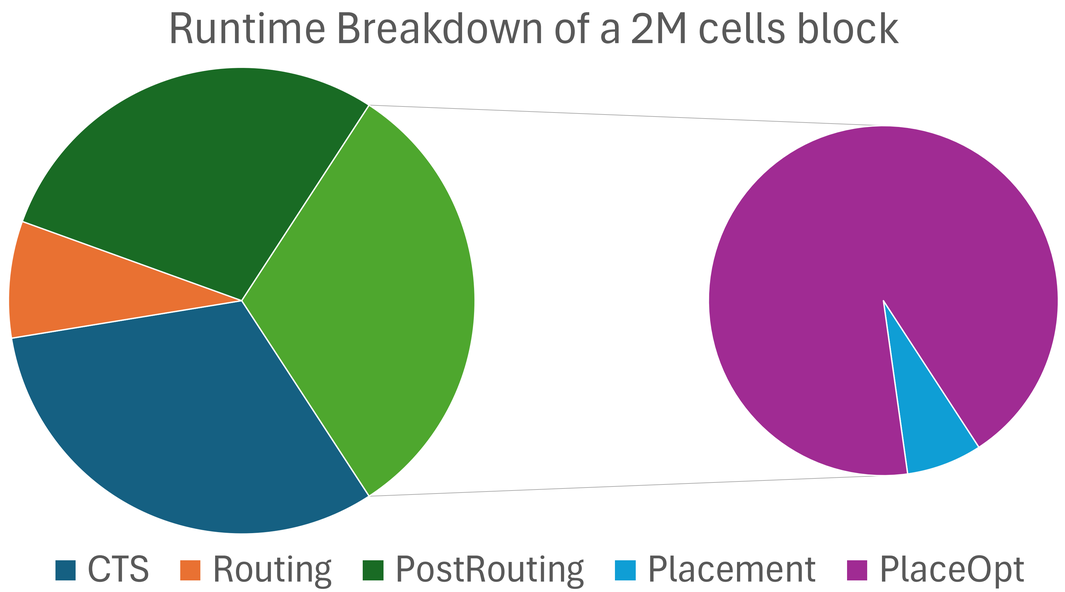
H. Ren et al, Nonlinear Placement for VLSI using TensorFlow on GPU, GTC'18; DREAMPlace, DAC'19

Optimization is the Main Scalability Challenge

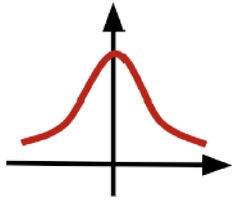
Optimization (gate sizing, buffering, etc. for better PPA) consumes the majority of the runtime in each major PD step

Why? Numerous heuristics to push for PPA limit, incremental timing, and routing updates

Is it optimal? Certainly not



Generative Model For Optimization



Auto Regressive (AR) Generative Model : $\mathrm{P}(x_{t+1}|x_1, x_2, \ldots, x_t)$

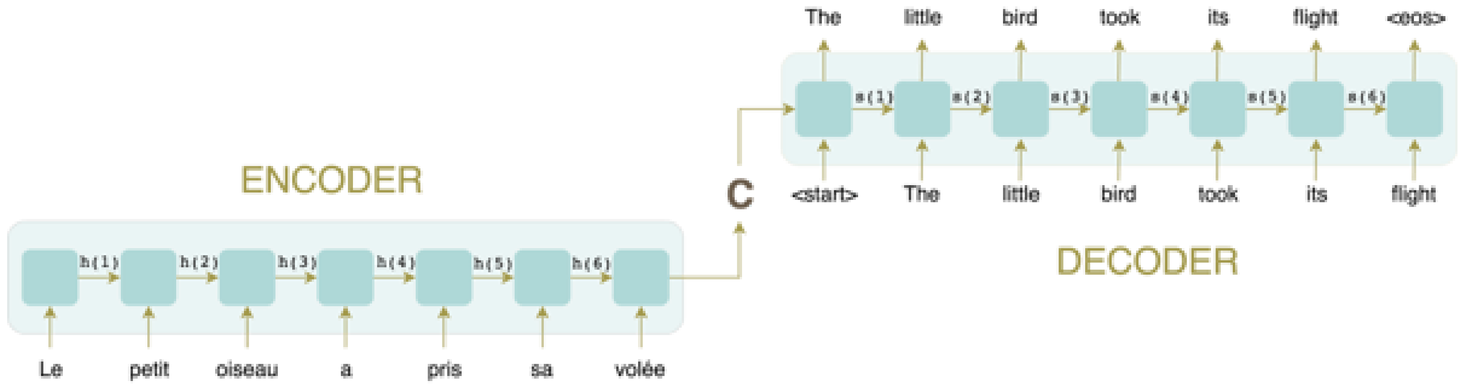
Consider the optimization variables as a sequence, generate optimized variable assignment one at a time

$$x_1 \rightarrow x_2|x_1 \rightarrow x_3|x_1, x_2 \ldots \rightarrow x_N|x_1, x_2, \ldots, x_{N-1}$$

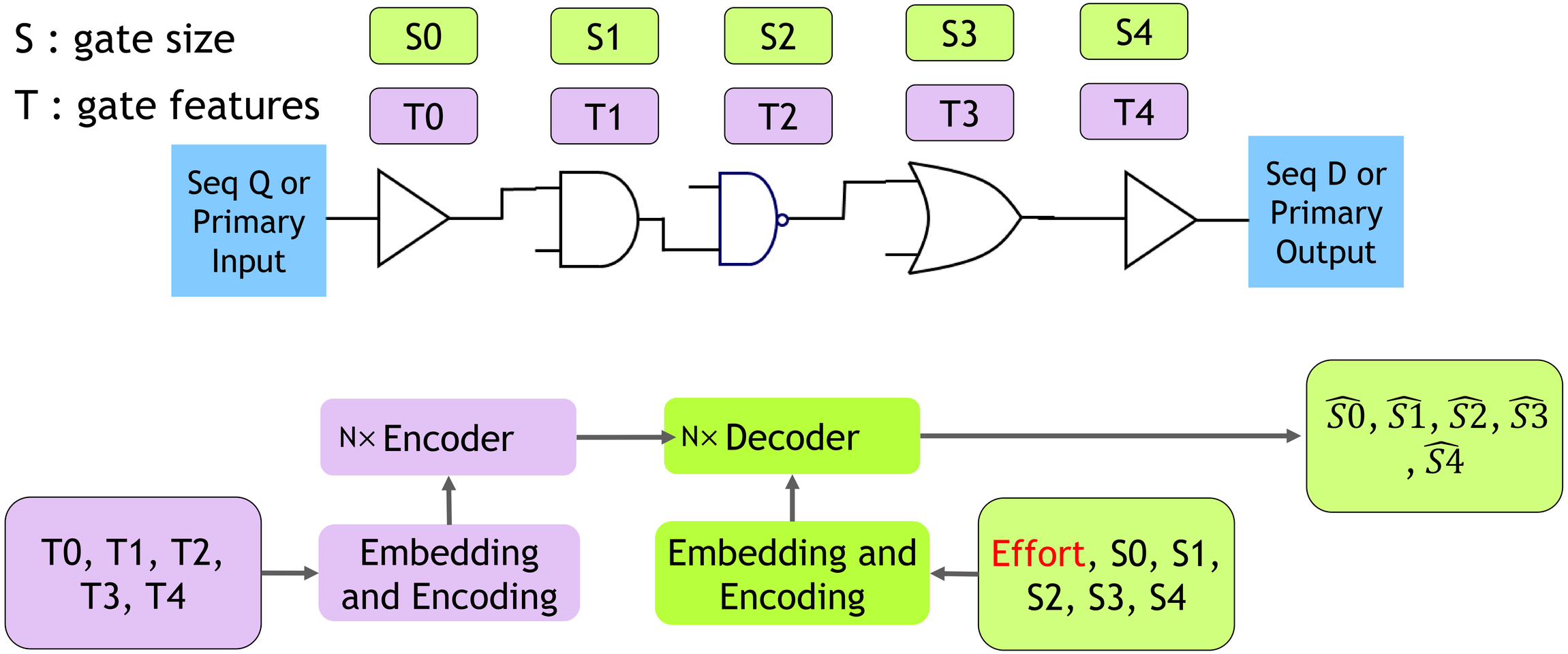
Conditioned on problem configuration C

$$x_1|c \rightarrow x_2|x_1, c \rightarrow x_3|x_1, x_2, c \ldots \rightarrow x_N|x_1, x_2, \ldots, x_{N-1}, c$$

Transformer can be used as conditional generative model



Gate Sizing with Conditional AR model



S. Nath et al. Transizer: A Novel Transformer-Based Fast Gate Sizer, ICCAD'22

Sizing Results

Accuracy is measured by comparing predicted sizes with true sizes： 88 – 92%

Compared to commercial tools, up to 1400X speedup with similar PPA

Generate initial condition alone can save big for runtime

Trained and inferenced on the same design, and results still not as good as tools

Name	Method	WNS	TNS	NVE	Power	Area	Speedup
Des1	*Init*	-0.507	-23.01	71	0.303	171	–
	Tool	0.0	0.0	0	0.304	172	1
	Ours	-0.03	-0.399	25	0.304	172	125X
Des2	*Init*	-0.991	-2062.09	7421	68.22	92.2K	–
	Tool	-0.105	-54.71	2539	69.13	92.3K	1
	Ours	-0.386	-54.81	917	68.22	92.2K	50X
Des5	*Init*	-1.360	-9870.29	33341	287.97	173.5K	–
	Tool	-0.551	-295.64	2920	287.98	174K	1
	Ours	-0.467	-344.13	5413	288.09	174K	1440X

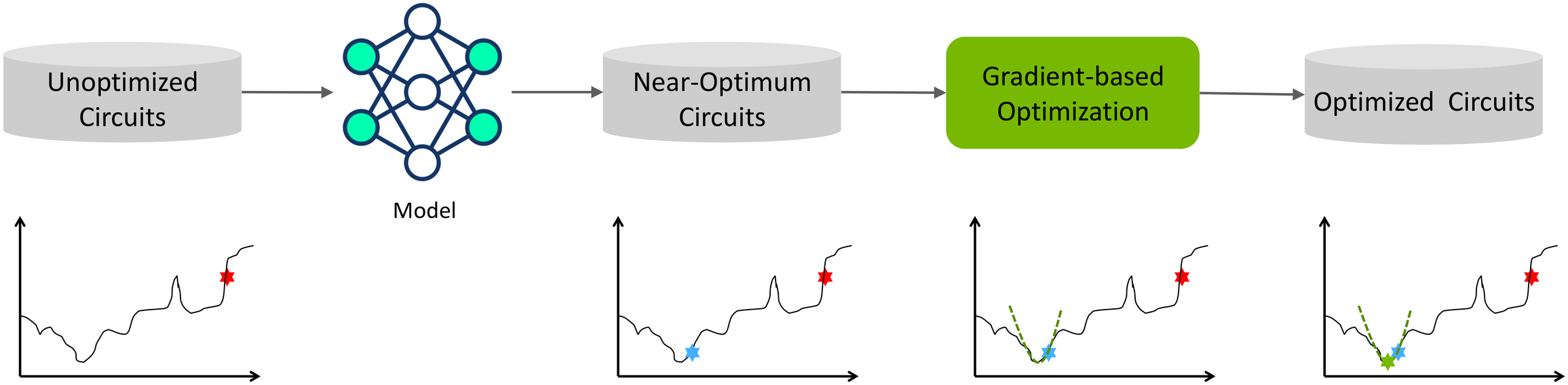
AI Optimization Method Comparison

	Advantage	Issues
Sequential Model-based Optimization (SMBO)	• Do not require gradients of the functions • Easy to use, always work • Commercial success	• Limited optimization variables (10s) • Require repeated evaluations of functions (100s – 1000s)
RL	• Only needs reward evaluation, and requires no gradient from the reward • Apply to dynamic/stochastic problems	• Limited optimization variables (100s) • Require fast reward evaluation (secs) • Long training time (hrs to days) • Difficult to generalize
Generative AI	• No need to compute objective function and its gradient • Scalable to millions of variables on GPUs	• Require a lot of optimized data • Data collection challenge • **How to deal with outliers**
Gradient-based Method	• Grounded optimization • Scalable to millions of variables on GPUs	• Require differentiable objective functions • Require relaxation and approximation • **How to ensure global optimal**

Combine Gen AI and Gradient-Based Method

Leverage Gen AI model to generate a solution that is close to the global optimum

Build gradient-based optimization in the global optimum neighborhood to search for the final solution

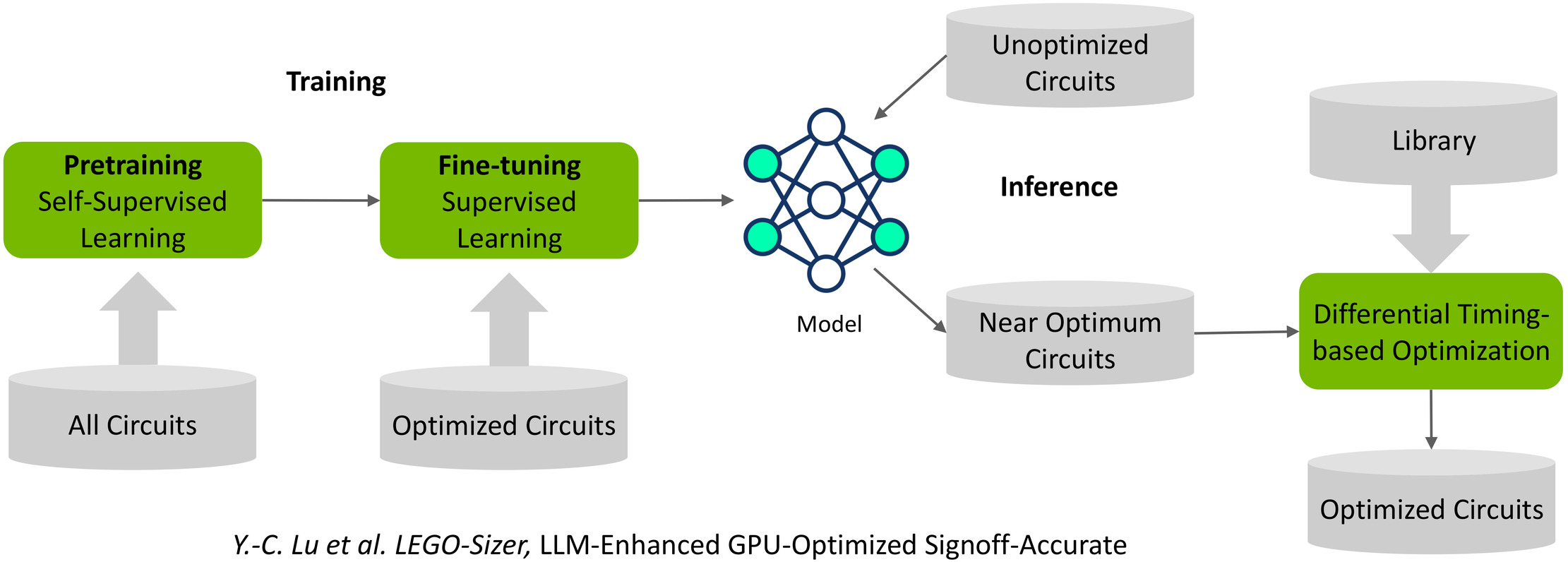


LEGO-Size

Encoder-only architecture predicts all gates on a path simultaneously

Similar to the pretraining/fine-tuning strategy, leverage unsupervised learning to learn the design representation and supervised learning to learn the optimization

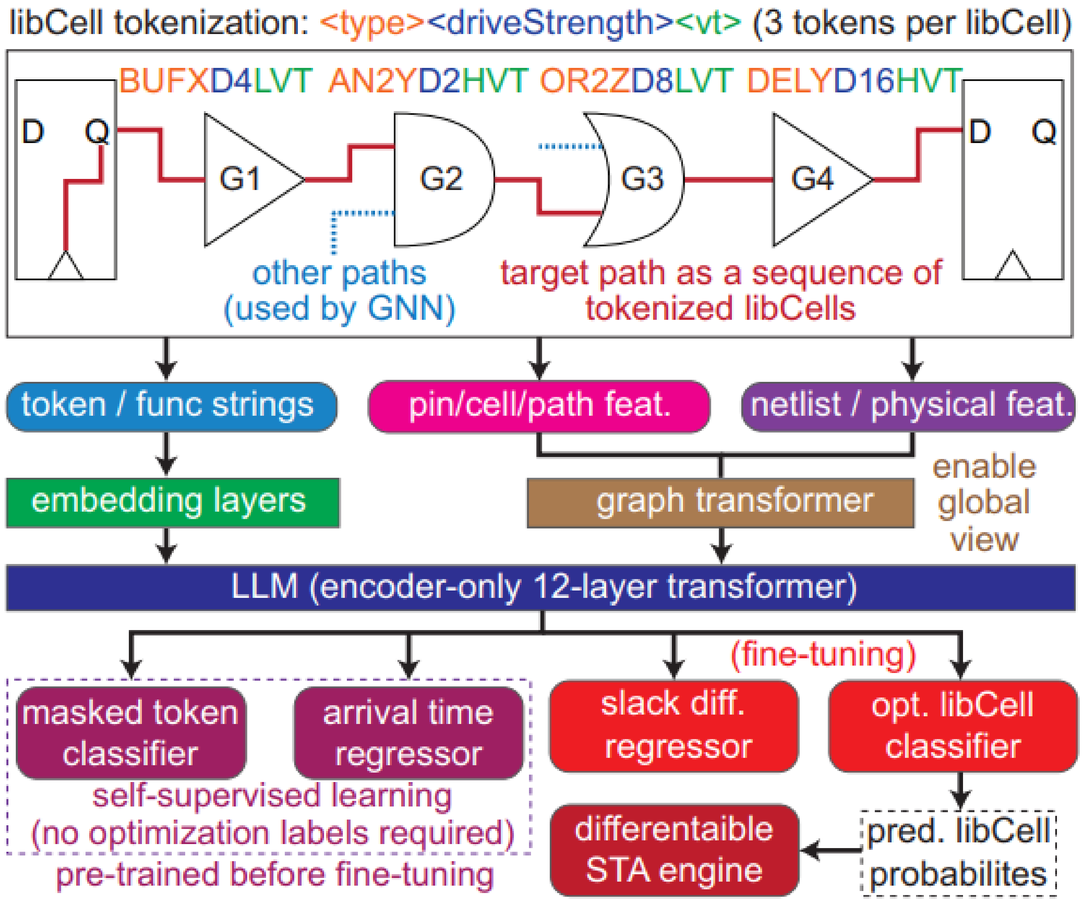
To fix the outliers after inference, run differential timing-based optimization



Y.-C. Lu et al. LEGO-Sizer, LLM-Enhanced GPU-Optimized Signoff-Accurate Differentiable VLSI Gate Sizing in Advanced Nodes, *ISPD'25*

Architecture

- Sophisticated embeddings to capture gate and netlist features
  - LLM-inspired tokenization for each gate
  - Timing table generated gate embeddings
  - Graph transformer to capture netlist features
- Pretraining tasks:
  - Masked token classifier
  - Arrival time regressor
- Fine-tuning tasks:
  - Slack difference regressor
  - Optimized libCell classifier
- Inference returns the initial probability of each libCell
- Differentiable STA engine built on PyTorch to optimize the probability of each libCell after inference



Results

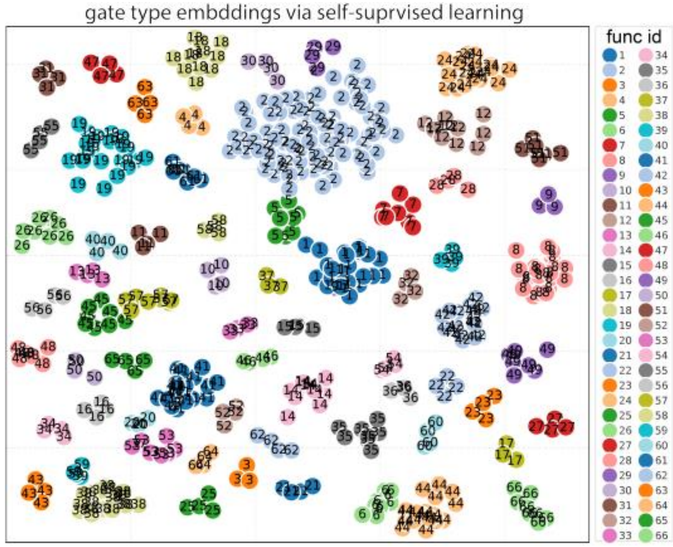
Pretrained on 22M timing paths over multiple 3nm blocks

Fine-tuned on ECO-optimized netlists

Inference on unseen designs from the fine-tuning

Better TNS/WNS optimization than ECO tools, >100X speedup

Learned embeddings show functional understanding



unseen design (# cells)	initial state (pre-optimized)				commercial signoff tool (32 threads)					LEGO-Size (GT + LLM + Differentiable STA)				
	WNS	TNS	#vio. EPs	total power	WNS	TNS (goal)	#vio. EPs	total power	speed-up	WNS	TNS (goal)	#vio. EPs	total power	speed-up
block1 (1.8M)	-0.066	-17.46	2410	238.6	-0.028	-1.953	271	238.8	1.00	-0.034	-1.237 (-36.7%)	184 (-47.3%)	238.8	125x
block2 (1.3M)	-0.041	-30.38	5410	293.4	-0.038	-7.82	1201	293.7	1.00	-0.035	-5.47 (-30.0%)	943 (-27.4%)	293.9	111x
block3 (1.2M)	-0.054	-10.03	1539	119.7	-0.032	-5.37	906	119.8	1.00	-0.033	-4.32 (-19.5%)	812 (-10.4%)	119.8	100x
block4 (1.5M)	-0.102	-59.25	6955	248.5	-0.097	-24.77	1912	249.1	1.00	-0.083	-20.36 (-17.8%)	1658 (-15.3%)	249.3	119x
block5 (1.4M)	-0.072	-11.33	1525	127.2	-0.038	-6.68	620	127.5	1.00	-0.036	-4.85 (-27.4%)	529 (-17.2%)	127.5	114x

Detailed Comparison

We only perform gradient descent on the cells that are predicted to be different by LLM

i.e., for cells whose argmax predictions are same as initial size, we do not perform GD on them

	Total	reg->reg	in->reg	reg->out	in->out
WNS	-0.834	-0.054	-0.206	-0.834	0.000
TNS	-1708.678	-10.026	-6.283	-1692.369	0.000
NUM	7393	1539	83	5771	0

initial

	Total	reg->reg	in->reg	reg->out	in->out
WNS	-0.834	-0.032	-0.204	-0.834	0.000
TNS	-1703.861	-5.374	-6.118	-1692.369	0.000
NUM	6758	906	81	5771	0

ECO Tool-optimized

Setup violations

	Total	reg->reg	in->reg	reg->out	in->out
WNS	-0.834	-0.035	-0.206	-0.834	0.000
TNS	-1705.311	-6.662	-6.280	-1692.369	0.000
NUM	7230	1376	83	5771	0

Model-only “predicted” @ F1 0.87

Setup violations

	Total	reg->reg	in->reg	reg->out	in->out
WNS	-0.834	-0.033	-0.205	-0.834	0.000
TNS	-1702.863	**-4.318**	-6.181	-1692.369	0.000
NUM	6666	**812**	83	5771	0

LEGO: Model + Gradient-based Optimization

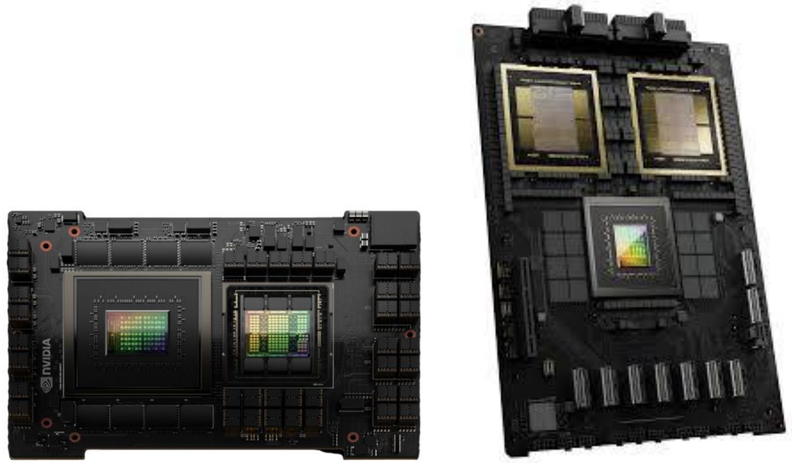
New EDA Computing Paradigm

Hardware determines software









GH and GB Superchips

Custom

Single thread CPU algorithms

Parallel CPU algorithms

Parallel CPU algorithms

GPU accelerated algorithms

Generative AI

