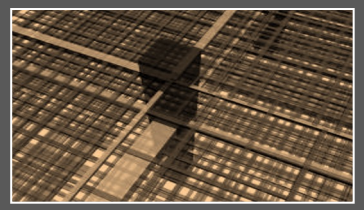
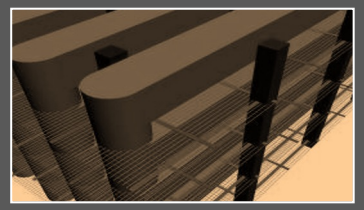
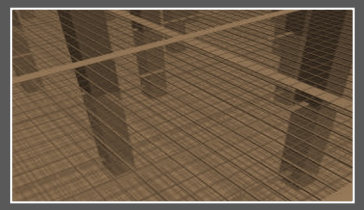
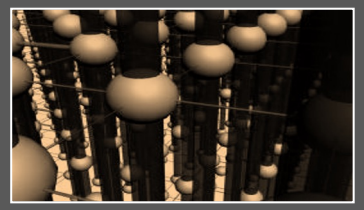
FastTuner: Transferable Physical Design Parameter Optimization using Fast Reinforcement Learning

Hao-Hsiang Hsiao, Yi-Chen Lu, Pruek Vanna-Iampikul, Sung Kyu Lim
Georgia Institute of Technology, Computer Aided Design Lab

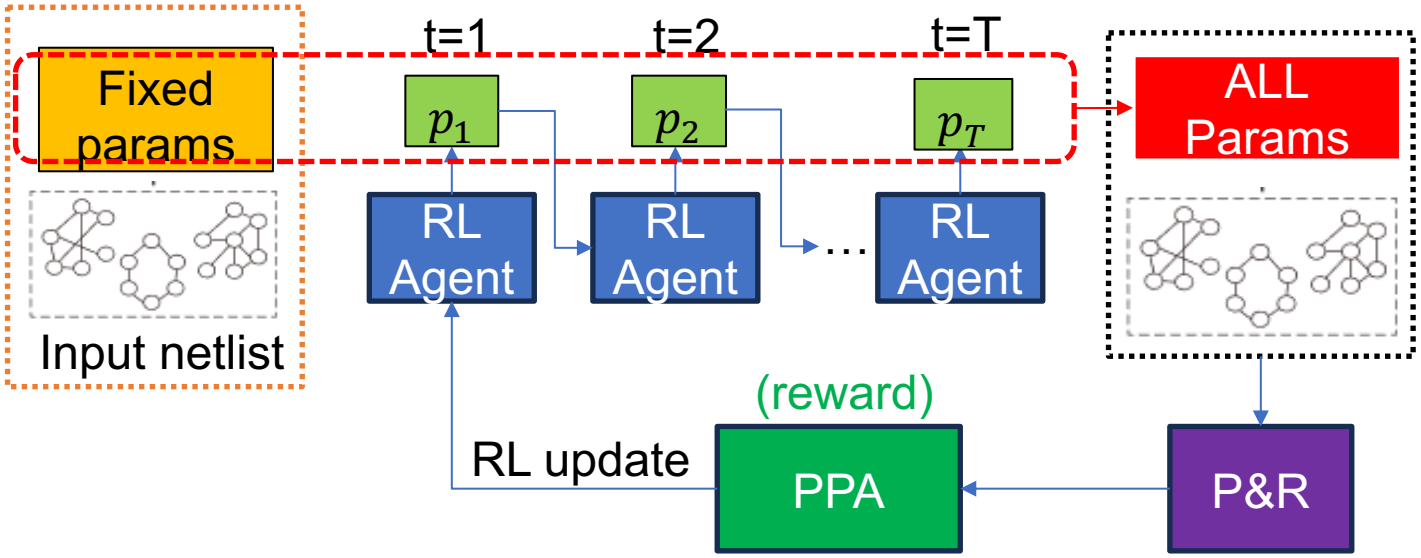
Motivation

- **Physical Design Parameter Optimization**
  - $\min_{p \in \mathcal{P}} f(p)$ where $f : \mathcal{P} \to \mathbb{R}$ is PPA, and $\mathcal{P}$ is the parameter space.
  - Design space is large => impossible to enumerate
  - Long Turnaround Time (TAT) for the function evaluation
  - Objective function PPA is non-differentiable (cannot compute gradient)

- **FastTuner**
  - Reinforcement Learning to **directly** optimize the non-differentiable PPA objective efficiently
  - Transfer learning with Graph Neural Network to leverage design knowledge and generalize across designs
  - PPA estimator provides instantaneous reward estimation
  - Encoder-Decoder architecture enables selectively tune a subset of parameters

Reinforcement Learning Formulation

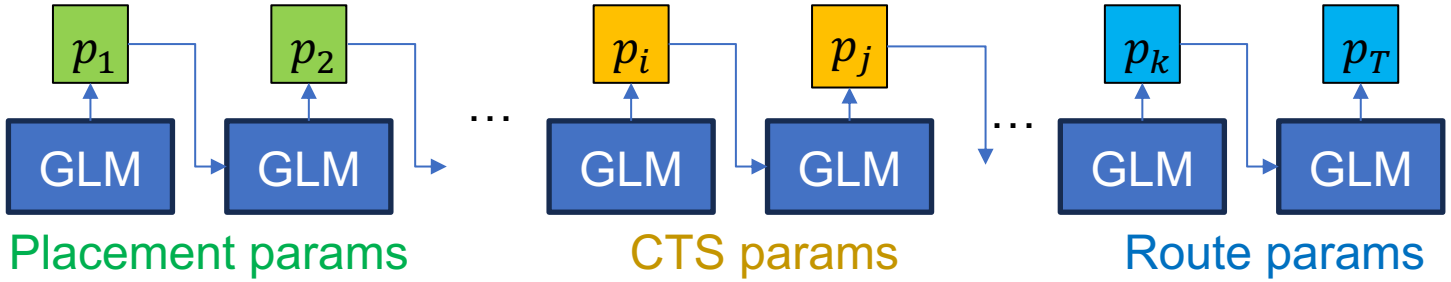
- State (s_t):
  - $h_\theta(s_1, s_2, \ldots, s_{t-1})$: configurations have been tuned from time steps 1 through $t-1$
  - GNN encoded netlist features
- Actions (a_t): valid values that can be selected for the parameter p_t
- Reward (r_t): $r_T = \text{postroute PPA}$
- Trajectory (τ): complete sequence of parameter selections from time step t = 1 to $t = T$



Sequential Optimization with Generative Language Models

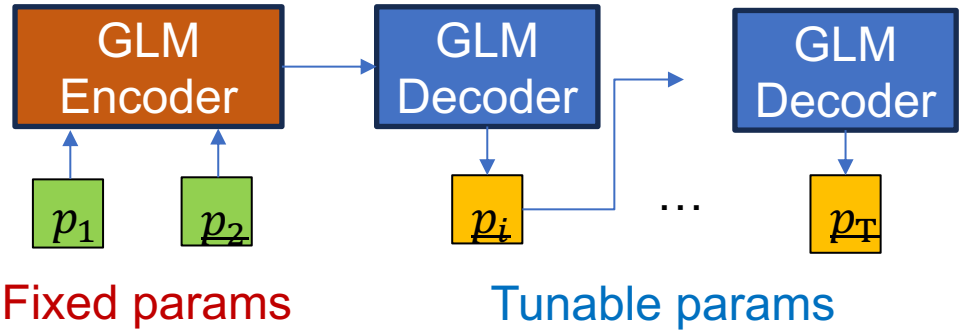
- We formulate parameter optimization as a sequential text generation task
  - Parameters are encoded as a sequence of text tokens
  - The Generative Language Models (GLM) Transformer predicts the best parameter setting for each parameter, based on the previous selections in the sequence

- Advantage:
  - Contextual Awareness: Autoregressive predict using all previous parameters for context, leading to informed predictions
  - Flexibility and Scalability: Handle variable-length sequences and parameter spaces through text-based generation
  - Rich Representation: GLMs encode complex parameter spaces in a versatile textual format



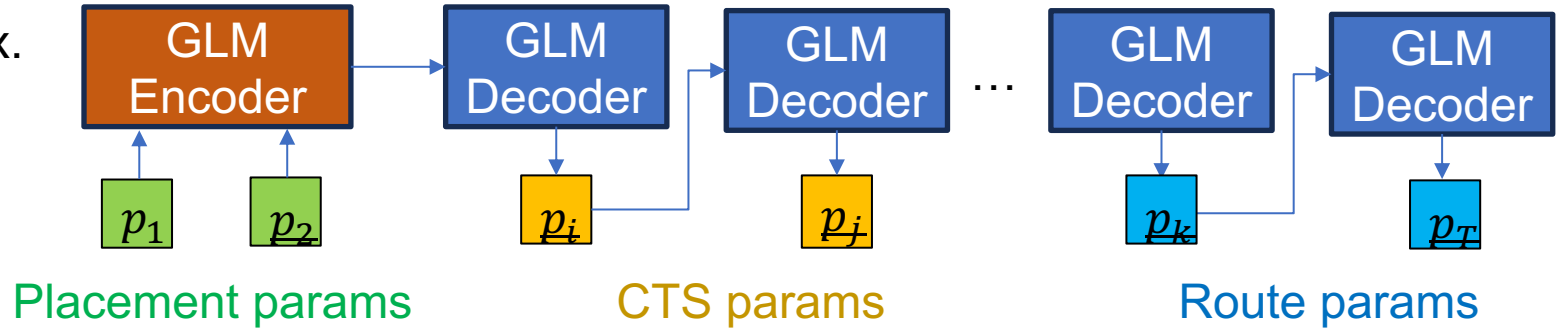
Tuning a Subset of Parameter

- Tuning Parameter as text completion
- Encoder:
  - encodes fixed parameters into contextual feature
- Decoder:
  - tunes selected parameters



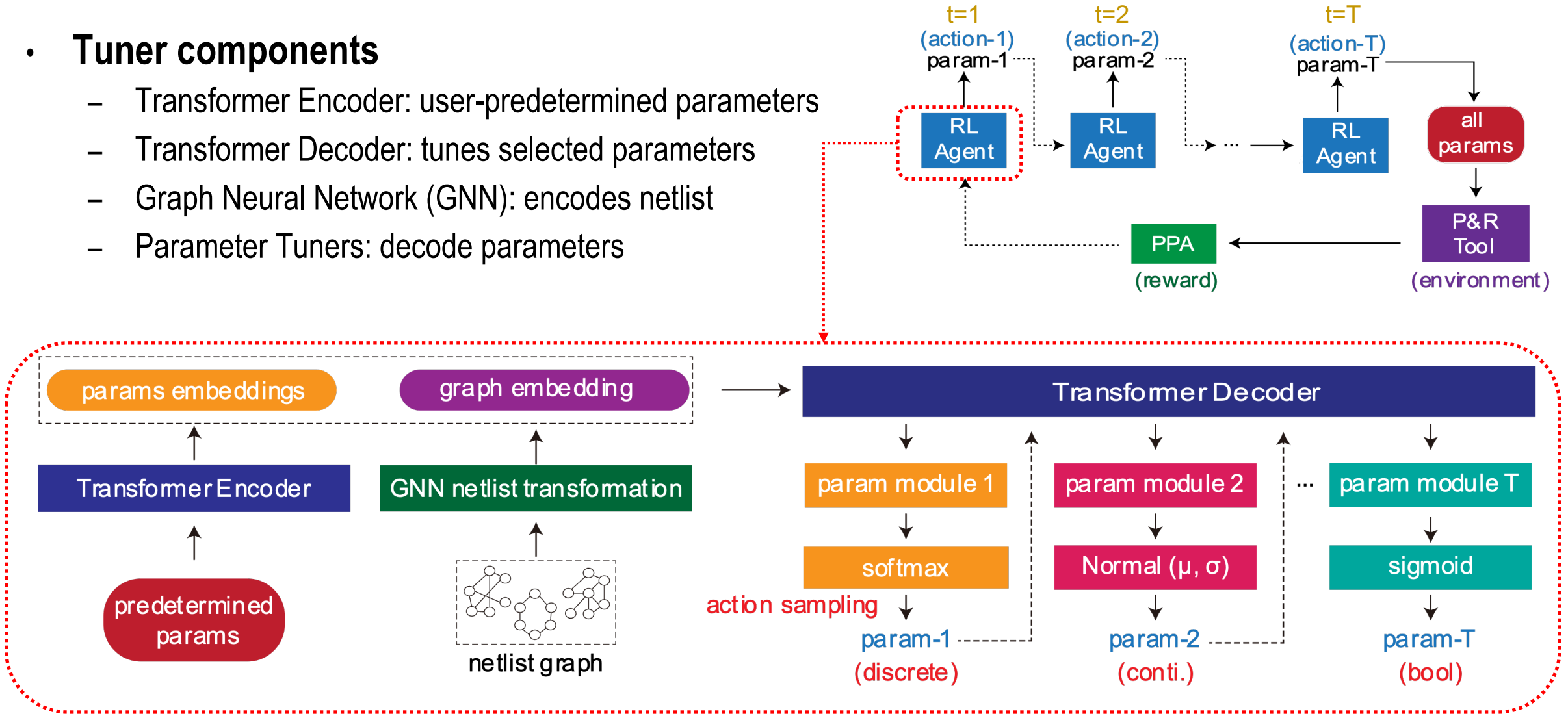
Encoder	Decoder
∅	Placement + CTS + Routing
Placement	CTS + Routing
Placement + CTS	Routing

Ex.



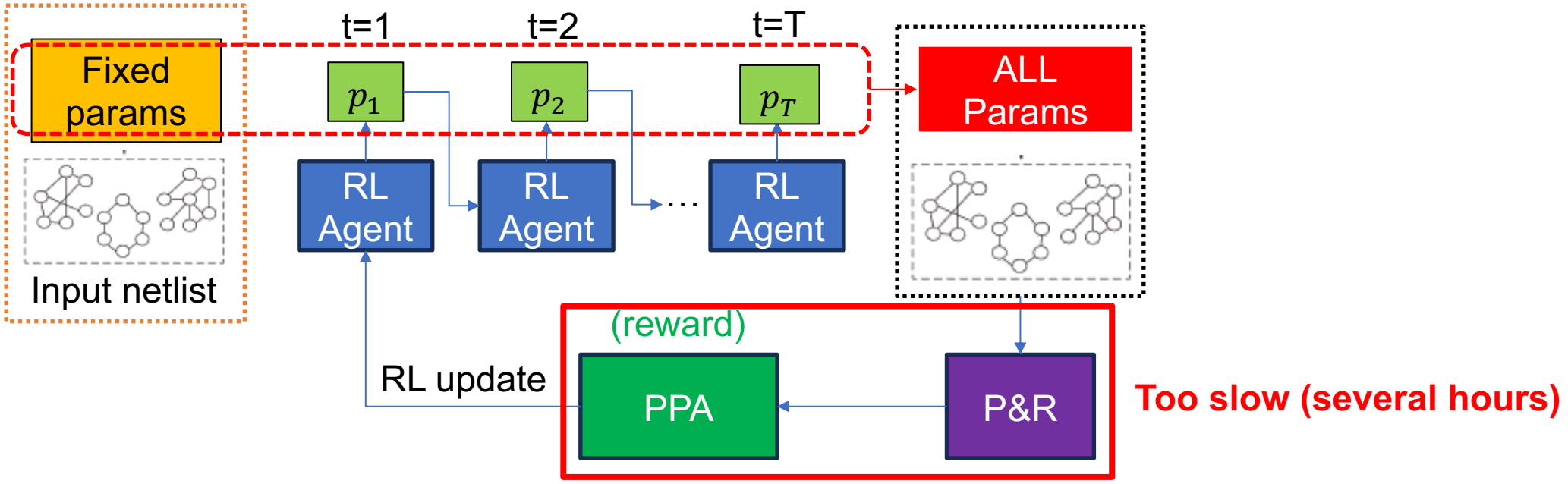
Model Architecture

- **Tuner components**
  - Transformer Encoder: user-predetermined parameters
  - Transformer Decoder: tunes selected parameters
  - Graph Neural Network (GNN): encodes netlist
  - Parameter Tuners: decode parameters



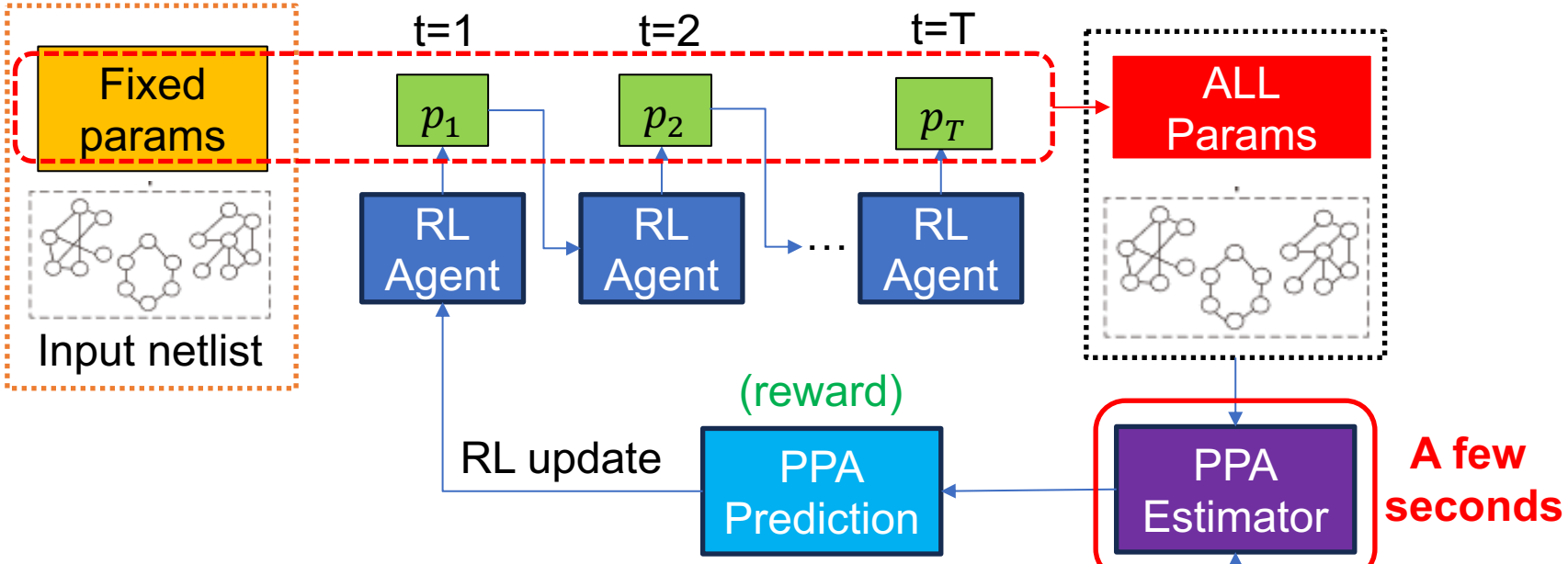
PPA Estimator

- Problem: the PPA feedback is slow and expensive
- Idea:
  - Learn an estimator to predict reward, use the estimation for control
  - Training time reduced from hours to minutes

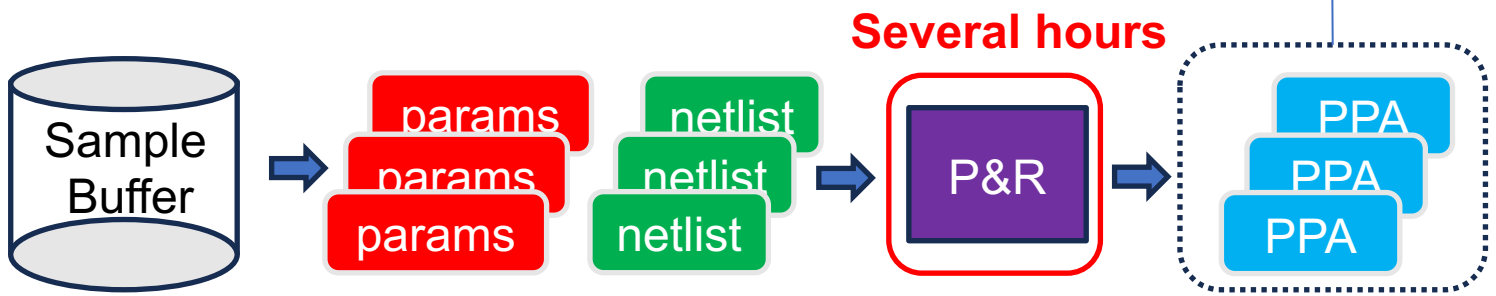


▲High-level views of our initial framework

FastTuner Framework



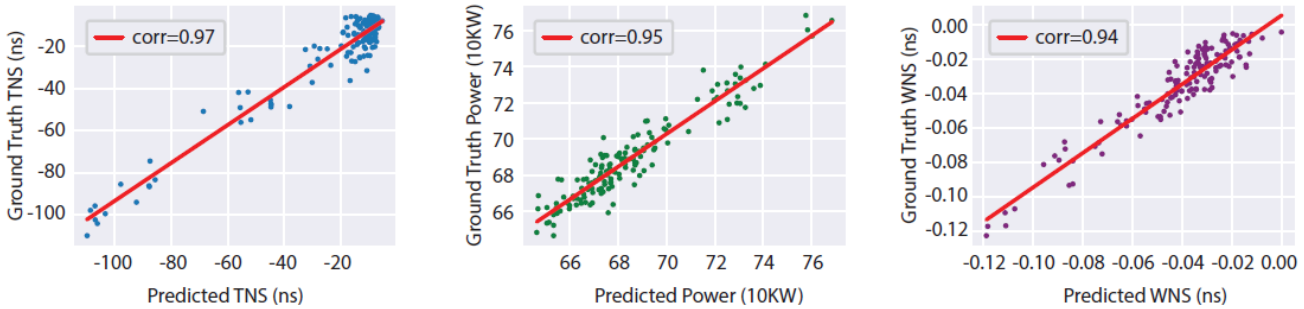
▲ Online Tuning: uses RL to optimize parameter



▲Offline Training: trains an PPA estimator offline as an ICC2 surrogate

PPA Estimator

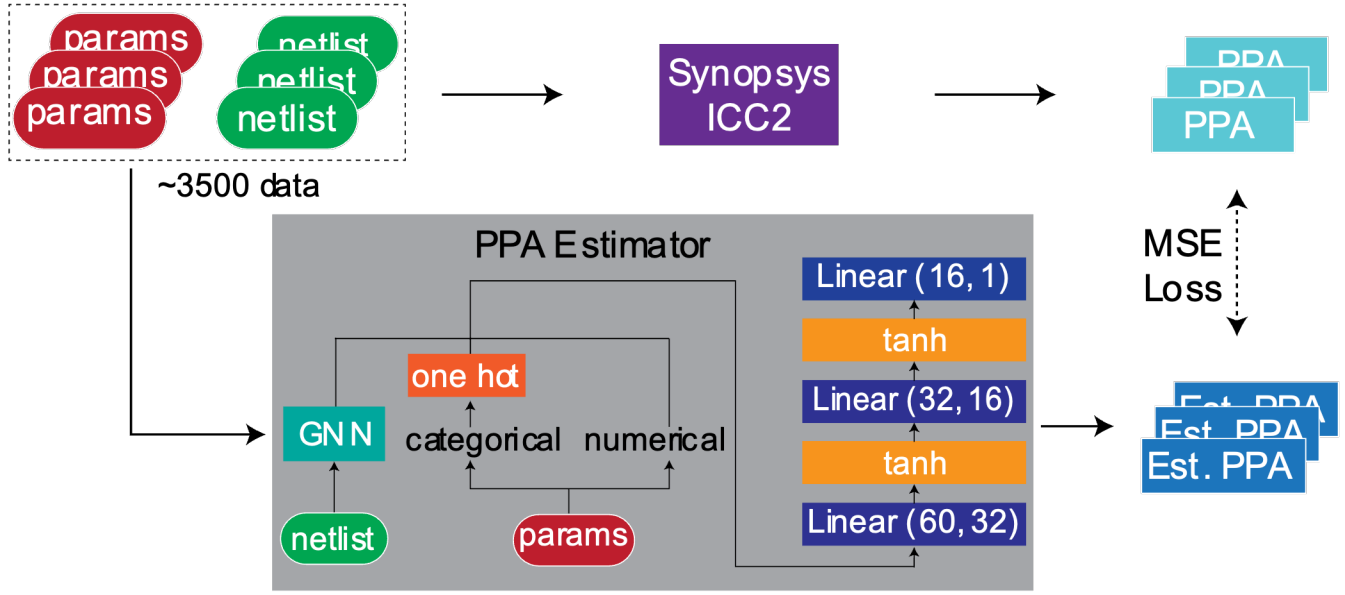
- Train an estimator **offline** as an ICC2 proxy
- Input
  - Parameter configurations
  - Netlist graph
- Output
  - Post-route PPA



▲Correlation analysis of our PPA estimator's predictions on the AES benchmark (test set)

Table 3: Prediction results of our PPA estimator on the validation sets of seven designs. "CC" denotes the Pearson correlation coefficient.

designs	TNS CC	Power CC	WNS CC
AES	0.97	0.95	0.94
DMA	0.91	0.93	0.90
LDPC	0.95	0.94	0.93
ECG	0.94	0.95	0.90
VGA	0.95	0.92	0.91
Commercial CPU	0.92	0.93	0.90
Rocket	0.90	0.91	0.90



▲Detail Architecture of the PPA estimator

Experimental Setup

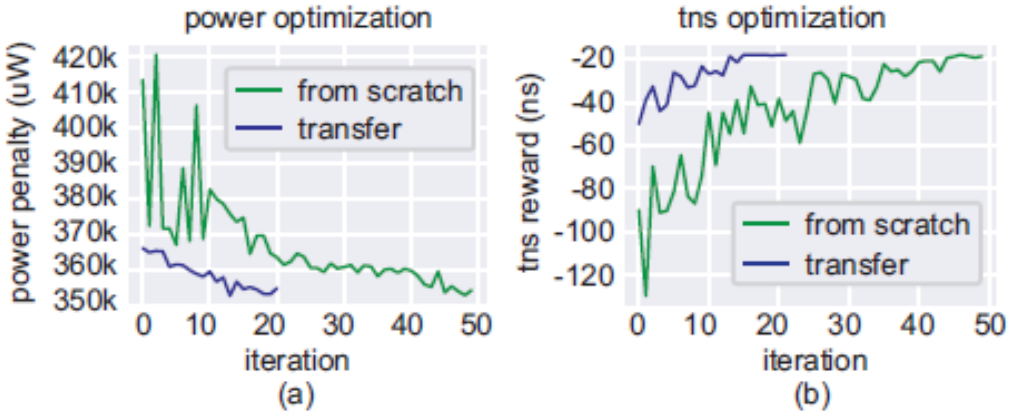
- Design tool: Synopsys ICC2
- Learning Framework: Pytorch, Pytorch Geometric
- Benchmark: 7 industrial benchmarks (TSMC 28nm)
- Experiment
  - Transfer Learning Results
  - Optimization Result Comparison vs. SOTA
  - Tuning a Subset of Parameters

Parameter	type	dims or ranges
place.coarse.target_routing_density	float	[0.7, 0.9]
place.coarse.max_density	float	[0.7, 0.9]
place_opt.initial_place.buffering_aware	bool	2
place_opt.initial_drc.global_route_based	int	[0, 1]
placement.aspect_ratio	float	[0.5, 1.5]
ccd.max_prepone	float	[-50 ps, 50 ps]
ccd.max_postpone	float	[-50 ps, 50 ps]
ccd.timing_effort	enum	3
cts.max_skew	float	[0.01, 0.2]
cts.max_fanout	float	[50, 250]
cts.max_buffer_density	float	[0.3, 0.8]
cts.max_latency	float	[0, 1]
route.common.rc_driven_setup_effort_level	enum	4
route.global.effort_level	enum	5
route.global.crosstalk_driven	bool	2
route.global.timing_driven	bool	2
route.global.timing_driven_effort_level	enum	2
route.track.crosstalk_driven	bool	2
route.track.timing_driven	bool	2
route.detail.optimize_wire_via_effort_level	enum	4
route.detail.timing_driven	bool	2
route_opt.flow.enable_power	bool	2
route_opt.flow.enable_irdrivenopt	bool	2

▲ Selected parameters

Transfer Learning Result

- FastTuner leverages previous training experiences for faster adaptation across various designs
- Transfering a pre-trained FastTuner model to an unseen netlist achieves comparable optimization results at a significantly faster convergence rate



▲Learning Curves of training from scratch vs. transfer learning

metrics	tool auto	aco	bo	FastTuner zero-shot
LDPC				
power (10^5uW)	2.82	2.66	2.58	2.6
tns (ns)	-150.2	-65.2	-74.77	-66.68
wns (ns)	-0.22	-0.11	-0.1	-0.12
pdp (10^5W*ns)	2.56	2.35	2.31	2.33
VGA				
power (10^5uW)	4.01	3.81	3.68	3.7
tns (ns)	-88.26	-50.7	-36.54	-46.2
wns (ns)	-0.38	-0.2	-0.16	-0.2
pdp (10^5W*ns)	2.89	2.68	2.64	2.66

▲Zero-shot FastTuner inference results
aco: ant coony opt.; bo: Bayesian opt.

Optimization Result

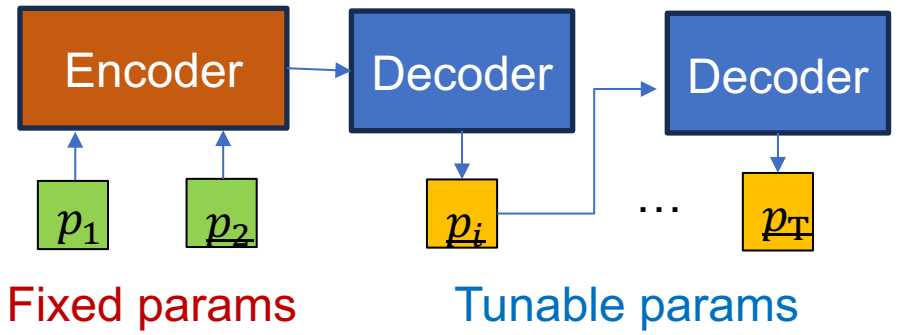
- Our approach demonstrates a better optimization result compared to SOTA Bayesian Optimization (BO) and Ant Colony Optimization (ACO) in terms of runtime and quality

metrics	tool auto	ACO	BO	FastTuner
LDPC				
power (10^5uW)	2.82	2.66	2.58	2.50 (11.35%)
tns (ns)	-150.20	-65.21	-74.77	-32.52 (78.35%)
wns (ns)	-0.22	-0.11	-0.10	-0.08 (64.38%)
pdp (10^5W*ns)	2.56	2.35	2.31	1.99 (22.45%)
runtime (#tool runs/ hours)	1/1.2	40/48	35/42	1/1.2
DMA				
power (10^5uW)	1.52	1.43	1.40	1.37 (10.16%)
tns (ns)	-96.67	-52.24	-29.13	-25.74 (73.38%)
wns (ns)	-0.21	-0.11	-0.13	-0.12 (42.86%)
pdp (10^5W*ns)	5.08	4.66	4.49	4.25 (17.32%)
runtime (#tool runs/ hours)	1/0.4	30/12	30/12	1/0.4

▲improvement: (%) compared with tool auto

Tuning a Subset of Parameters

- Our encoder-decoder framework enables selective tuning of specific parameter subsets
- FastTuner effectively optimizes the chosen subset conditioned on the user-determined parameters



▲ encoder-decoder framework

metrics	tool auto	1	2	3
LDPC				
power	6.71	5.85	5.58	5.94 (11.48%)
tns	-101.25	-46.26	-30.98	-28.74 (71.62%)
wns	-0.08	-0.12	-0.10	-0.03 (62.77%)
pdp	1.51	2.26	1.97	1.20 (20.30%)
VGA				
power	1.52	3.81	3.60	1.37 (10.16%)
tns	-96.67	-59.06	-28.35	-25.74 (73.38%)
wns	-0.21	-0.20	-0.17	-0.12 (42.86%)
pdp	5.08	2.64	2.35	4.25 (17.32%)

	Encoder (Fixed)	Decoder (Tuned)
1	Placement + CTS	Routing
2	Placement	CTS + Routing
3	∅	Placement + CTS + Routing

▲ 3 different scenarios we test

Conclusion

- Our main contributions
  - We propose a hybrid FastTuner framework that enables online RL tuning using offline-trained PPA estimators to significantly speedup the tuning process
  - Our framework is the first to utilize generative language model to optimize parameter and offers the flexibility to selective tuning a subset
  - FastTuner facilitates transfer learning to generalize across various design
  - Our methods consistently demonstrated superior results compared to SOTA approaches across different benchmarks and objectives
- Future work
  - More complex industrial designs and more advanced technology nodes
  - Optimization on different flow, e.g. 3D IC
  - Tackle objectives beyond PPA
  - Improve PPA proxies

Thank You for Listening! Q&A