ASIA SOUTH PACIFIC DESIGN AUTOMATION CONFERENCE
ASP-DAC 2025
30th Asia and South Pacific Design Automation Conference

PIRLLS: Pretraining with Imitation and RL Finetuning for Logic Synthesis

Guande Dong, Jianwang Zhai, Hongtao Cheng, Xiao Yang, Chuan Shi, Kang Zhao

{dongguande, zhaijw, zhaokang}@bupt.edu.cn

Beijing University of Posts and Telecommunications

Contents

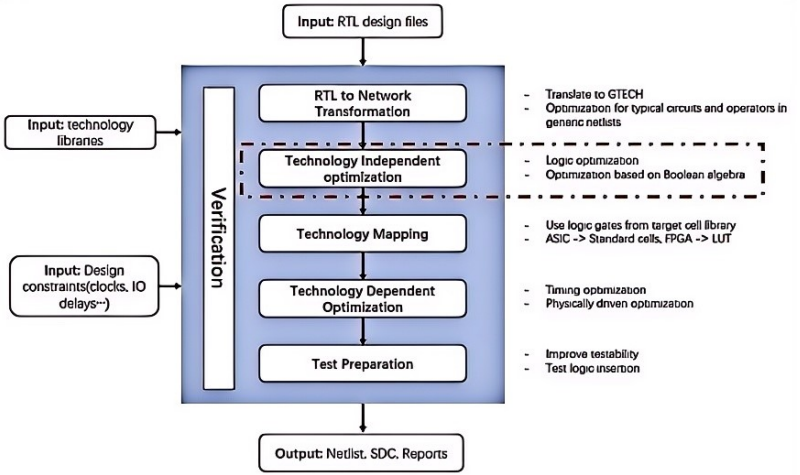
- **Introduction**
- **Preliminaries**
- **Motivation**
- **Framework**
- **Experiments**
- **Conclusion**

Introduction

Introduction

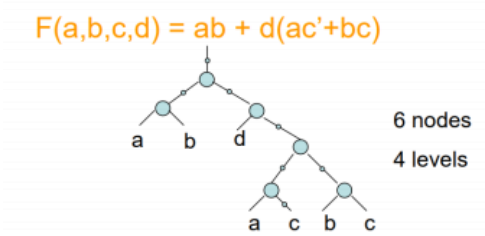
- Logic synthesis transforms a high-level circuit description at the Register-Transfer Level (RTL) into an optimized gate-level netlist.

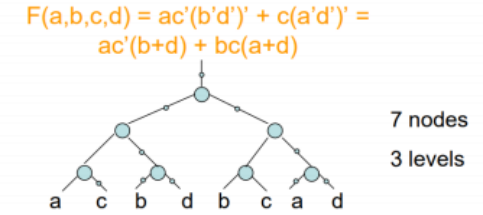
- Logic optimization directly affects technology mapping, determining the final netlist's area and delay.



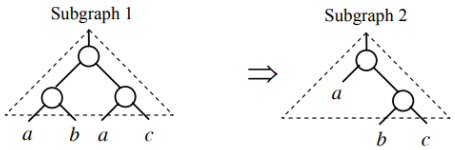
Introduction

- The popular open-source tool ABC [1] represents combinational logic using And-Inverter Graphs (AIGs).

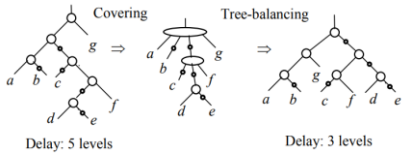




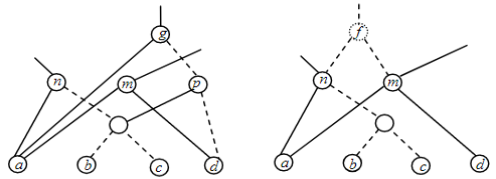
- ABC provides various logic optimization operators, such as:



(a) rewrite



(b) balance



(c) resub

[1]ABC: An academic industrial-strength verification tool,CAV 2010

Introduction

Challenges

- The exponentially growing search space makes design space exploration challenging. For a set $A = \{a_1, a_2, \ldots, a_n\}$ and flow length k, there are n^k possibilities.
- Different circuits have unique characteristics, requiring tailored optimization operators flows to achieve the best results.
- Challenges remain in representing AIG states and optimization flows, hindering the application of machine learning methods.

Introduction

To overcome the above challenges, many design space exploration methods have been proposed.

Related Work

- **DRiLLS constructs scalar features and integrates the A2C agent[1].**
- **GNNs capture AIG topology and combine it with historical decisions[2].**
- **Random Forests analyze feature importance for pruning[3].**
- **Context-based multi-armed bandit with Syn-LinUCB[4].**
- **Monte Carlo Tree Search with SynUCT[5].**

[1] K. Zhu,et al. “Exploring Logic Optimizations with Reinforcement Learning and Graph Convolutional Network,” in Proc.,MLCAD,2020
[2] Hosny, Abdelrahman,et al. “DRiLLS: Deep Reinforcement Learning for Logic Synthesis.” ASP-DAC ,2019
[3]Zhou, G,et al. “Area-Driven FPGA Logic Synthesis Using Reinforcement Learning.” ASP-DAC,2023
[4] F Liu, et al.”CBTune: Contextual Bandit Tuning for Logic Synthesis. ”DATE,2024
[5] Z. Pei *et al*. "AlphaSyn: Logic Synthesis Optimization with Efficient Monte Carlo Tree Search." ICCAD,*2023*

Introduction

Previous Solutions & Limitations

- Supervised learning-based methods rely on limited datasets and often suffer from poor prediction accuracy.
- Online reinforcement learning and bandit-based methods start training from scratch for each encountered circuit, lacking generalization capability.

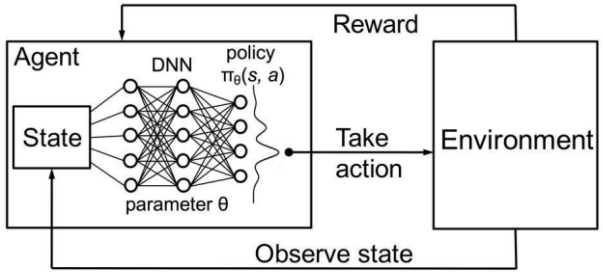
Therefore, an ideal policy should consider how to learn general offline knowledge from existing circuits and synthesis flows and fine-tune it for new target circuits.

Preliminaries

Preliminaries

Reinforcement Learning

- Reinforcement Learning (RL) enables an agent to learn optimal actions by interacting with an environment and receiving feedback in the form of rewards.
- The goal is to maximize cumulative rewards over time, adapting the policy based on observed outcomes.



Logic Synthesis
CAD Tool

Preliminaries

Imitation Learning

Imitation Learning is a machine learning approach that aims to learn policies by mimicking expert behavior. In imitation learning, the model learns to take appropriate actions in similar environments by observing expert demonstrations (state-action pairs).

Behavior Cloning (BC) is a specific method of imitation learning, similar to supervised learning. It frames the imitation problem as a classification or regression task, fitting the expert's state-action mapping to learn a policy.

Motivation

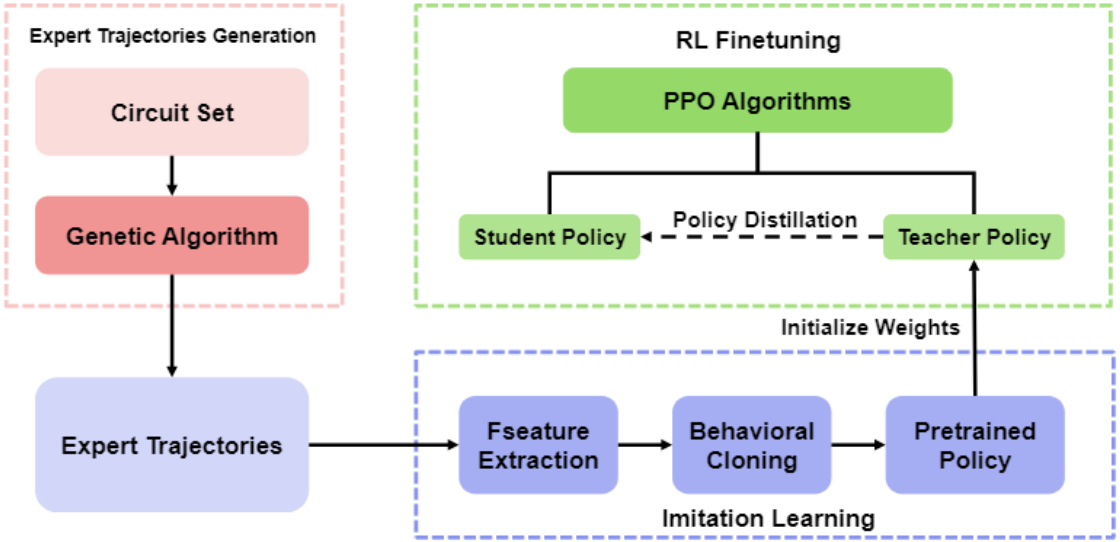
Simple experiment

Circuit	Best	Change	resyn2*2
s838	59.9	63.5	65
s838.1	61.6	67	70
C7552	337.2	355.9	382
c7552	348.5	365.3	380
C3540	209	218.2	228
c3540	212.8	215	230
C2670	111.9	118	130
c2670	190	192	211.125

Framework

PIRLLS Framework

- **Expert Trajectories Generation**
- **Imitation Learning**
- **RL Finetuning**



Expert Trajectory Generation

Use GA to generate high-quality synthesis flows as training data

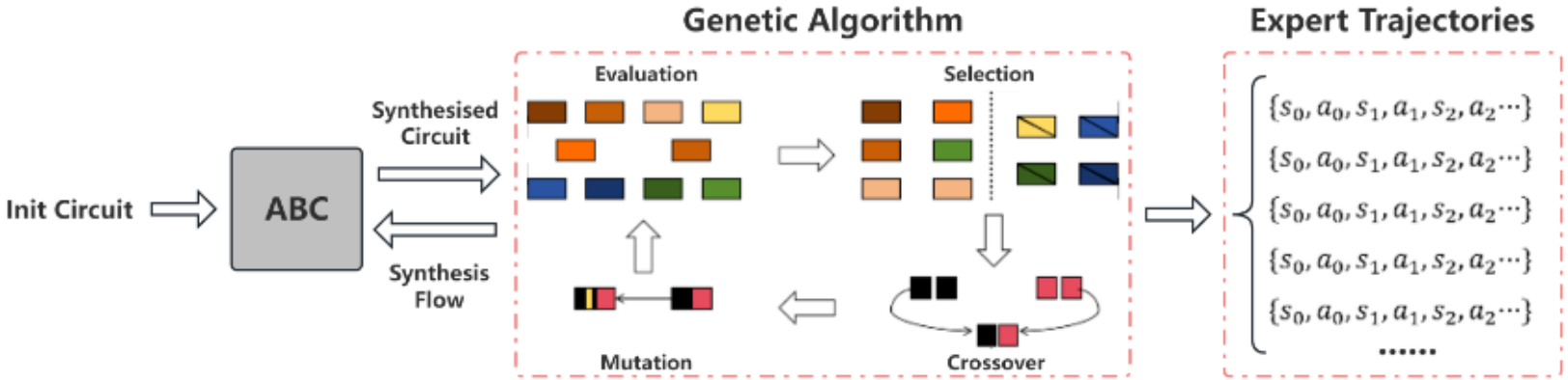
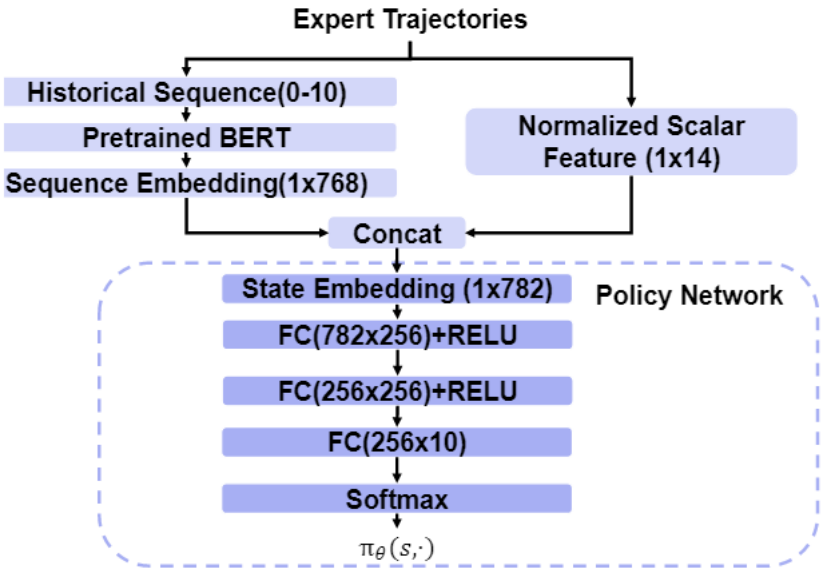


Figure 3: Schematic diagram of expert trajectory generation

Imitation Learning

Feature Extraction

Poilcy network



Scalar Features of AIG Circuits

ID	Feature	ID	Feature
1	Input	2	Output
3	Number of gates	4	Level
5	Width	6	Number of LUTs
7	Number of LUTs level	8	Avg of input node fan-out
9	Std of input node fan-out	10	Avg of and node fan-out
11	Std of and node fan-out	12	And_nodes percent
13	Not_nodes percent	14	And_node_reduction_percent

Imitation Learning

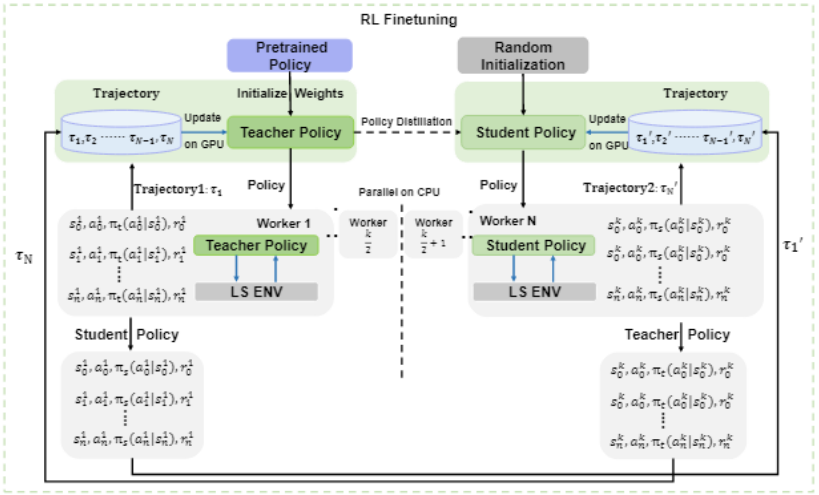
The policy, denoted as $\pi_\theta^{BC}(a_t|s_t)$ maps a given state s_t to an action a_t. Expert trajectories are represented as sequences of states and actions, $\tau = (s_0, a_0, \cdots, s_t, a_t)$

L_{NLP} is the negative log-likelihood of expert actions under the policy's predicted distribution:

$$L_{NLP} = -\sum_{i=1}^{N} \sum_{(s_t,a_t)\in\tau^{(i)}} \log \pi_\theta^{BC}(a_t|s_t).$$

RL Finetuning

- **Dual Policy Distillation**
  - Teacher Policy: High-quality trajectories for efficient learning
  - Student Policy: Random initialization for broader exploration
- **Multi-process parallel interaction**
- **Shared Critic Network**



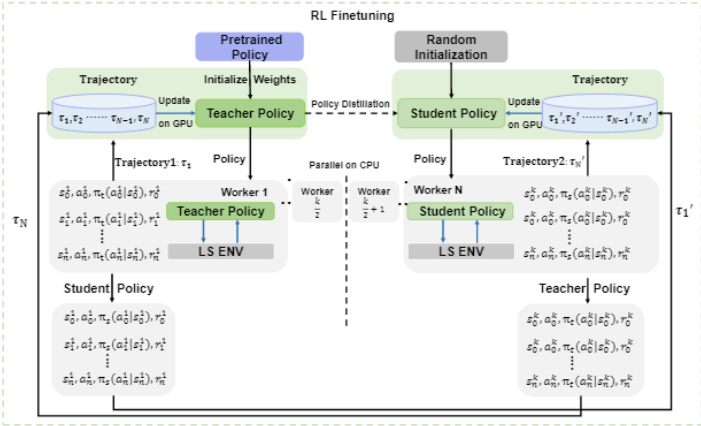
RL Finetuning

The **Student** Policy's loss

$$L_{\text{student}} = L_{\text{policy}} + L_{\text{value}} + \alpha L_{\text{KL}}, \qquad L_{\text{KL}} = \sum_{a \in A} \pi_s(a|s) \log \left(\frac{\pi_s(a|s)}{\pi_t(a|s)} \right).$$

The **Teacher** Policy's loss

$$L_{\text{teacher}} = L_{\text{policy}} + L_{\text{value}} + \beta L_{\text{Entropy}}, \qquad L_{Entropy} = -\sum_{i=1}^{N} \sum_{s_t \in \tau^{(i)}} \sum_{a \in A} \pi(a|s_t) \cdot \log \pi(a|s_t),$$



Evaluation

Evaluation

- **Training sets: ISCAS'85 , ISCAS'89 , ITC'99 , LGSynth'89 , LGSynth'91 , IWLS'93 , IWLS 2005 , and LEKO/LEKU benchmarks**
- **Testing sets : EPFL benchmark**
- **Baseline:DRILLS[1],RL4LS[2],CBtune[3],Alphasyn[4]**
- **ABC for logic synthesis optimization**

  Mapped to 6-LUTs using the ABC priority cuts mapper (command: if -a -K 6)
- **RL algorithm evaluated: PPO**
- **Train for 5k samples**

---

[1] Hosny, Abdelrahman,et al. “DRiLLS: Deep Reinforcement Learning for Logic Synthesis.” ASP-DAC ,2019
[2]Zhou, G,et al. “Area-Driven FPGA Logic Synthesis Using Reinforcement Learning.” ASP-DAC,2023
[3] F Liu, et al.”CBTune: Contextual Bandit Tuning for Logic Synthesis. ”DATE,2024
[4] Z. Pei *et al*. "AlphaSyn: Logic Synthesis Optimization with Efficient Monte Carlo Tree Search." ICCAD,*2023*

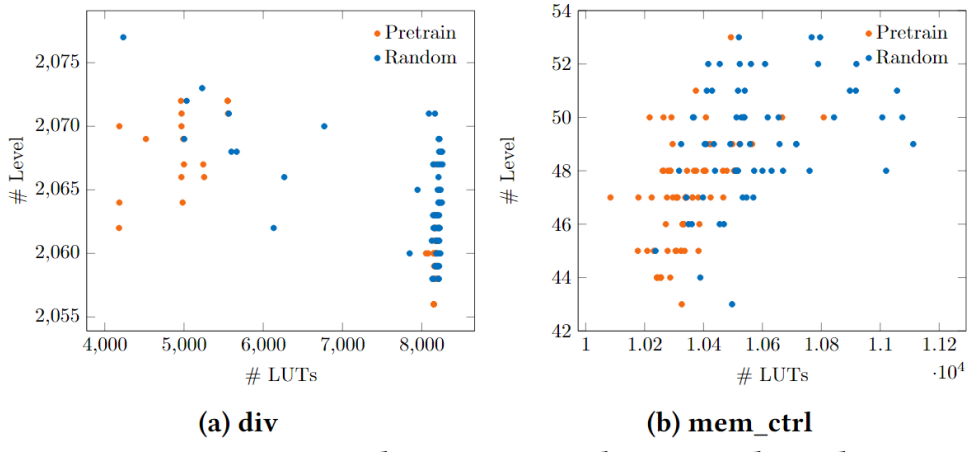
Evaluation

We achieved 5% better LUT optimization and a 2.4× speedup compared to the current best method

Benchmark	Initial	resyn2*2	DRiLLS [4]		RL4LS [6]		AlphaSyn [8]		CBTune [7]		Pretrain Only		PIRLLS	
	#LUTs	LUTs	#LUTs	RT(m)	#LUTs	RT(m)	#LUTs	RT(m)	#LUTs	RT(m)	#LUTs	RT(m)	#LUTs	RT(m)
max	721	719	694	32.58	687.8	54.34	680	5.7	684.25	6.01	688	**0.183**	**675.6**	2.49
adder	249	249	244	24.05	244	10.05	244	6.14	244	5.97	211	**0.184**	**196.7**	1.62
cavlc	116	118	112.2	26.02	111.3	3.22	**106**	5.35	111	2.37	115	**0.155**	107.6	1.64
ctrl	29	29	28	24.25	28	2.85	28	5.68	28	0.59	29	**0.156**	**27**	1.66
int2float	47	46	42.6	21.7	42.3	2.81	**39**	5.54	40	2.76	42	**0.165**	**39**	1.54
router	73	76	70.1	22.01	69.5	3.07	65	5.34	68.11	2.32	72	**0.16**	**61.3**	1.64
priority	264	220	133.4	23.32	142.9	5.9	135	5.83	138.86	3.41	163	**0.193**	**126.3**	1.58
i2c	353	320	292.1	25.17	289.32	7.55	280	6.22	283.11	3.61	299	**0.165**	**271.3**	1.85
sin	1444	1466	1441.5	51.15	1438	20.1	1438	6.77	1441.67	9.71	1450	**0.235**	**1434.75**	3.57
square	3994	3915	3889.4	130	3889	72.88	3877	8.72	3882.11	25.99	3889	**0.39**	**3849.5**	7.92
sqrt	8084	5127	4708	147.64	4685.3	196.15	4415	9.83	4607	36.51	4038	**0.47**	**3837**	17.81
log2	7584	7703	7583.6	198.6	7580.1	125.28	7580	11.78	7580	41.27	7584	**0.61**	**7469.6**	12.8
multiplier	5678	5713	5678	180.84	5672	187.81	5672	10.34	5679.75	29.08	5675	**0.45**	**5663.6**	10.57
voter	2744	1828	1834.7	84.43	1678.1	330.48	**1537.4**	7.84	1682.25	11.46	1688	**0.28**	1579	3.7
div	23864	8197	7944.7	259.75	7807.1	482	6650.1	11.88	4180.91	25.58	4124	**0.48**	**3963.3**	17.52
mem_ctrl	11631	11459	10527.6	229.33	10309.7	1985.84	**9513.2**	10.99	10242.57	45.81	9838	**0.64**	9547.5	17.39
Average	4179.67	2949.06	2826.40	92.55	2792.20	218.14	2641.23	7.75	2555.84	15.78	2494.06	**0.31**	**2428.06**	6.58
Ratio	1.72	1.21	1.16	14.07	1.15	33.15	1.09	1.18	1.05	2.40	1.03	**0.05**	**1.00**	1.00

Evaluation

- Comparison between exploration based on pretraining and random exploration from scratch.



(a) div (b) mem_ctrl

Table 4: LUT Results of Ablation Study

Benchmark	RL	RL+His	RL+BERT		Pretrain+RL+BERT	
	Last	Last	First	Last	First	Last
max	688	691	716	688.1	711	**685.6**
adder	203.1	201	216.6	200.7	211.1	**198.1**
cavlc	111.1	113.9	118.1	111.1	113.9	**110.2**
ctrl	28	28.1	28.8	28	28.2	**27.9**
int2float	43.9	43.4	43.9	**41.4**	43.4	41.7
router	66.5	65.4	70.5	64.4	66.8	**64.2**
priority	142.9	140.7	170.1	135.2	152.5	**127.9**
i2c	289	285.45	299.2	**272.9**	297	274.7
sin	1441.3	1439.7	1452	1440	1445.7	**1438.2**
square	3889	3886.7	3894.6	3862	3889	**3851**
sqrt	4692	4686	4249.7	3870.7	3936	**3847.4**
log2	7489.1	7584	7599.3	7479.1	7548.4	**7471**
multiplier	5687.6	5692	5687.6	5671.4	5692	**5661.6**
voter	1659	1672	1787	1632.4	1716.3	**1611.2**
div	4543	4436	4828	4386.3	4109.7	**4035.9**
mem_ctrl	10291.7	10267.1	10470.5	10274.6	10299.6	**9618.4**
Average	2579	2577	2601.9	2509.8	2516.2	**2441.5**

Conclusion

- We propose PIRLLS, a two-stage learning framework for logic synthesis and optimization.
  - Pretrain the exploration with imitation learning on expert trajectories, leveraging offline knowledge from existing data.
  - Finetune the pretrained policy on target circuits using reinforcement learning, customizing optimization for specific needs.
- PIRLLS surpasses state-of-the-art methods across various metrics, enhancing optimization quality and achieving significant speedup.

THANK YOU!