

MetaSyn: A Meta-Reinforcement Learning Framework with Multimodal Circuit Representation for Adaptive Logic Synthesis

Shukai Liu^{1,2}, Ruoyan Liao³, Siyu Wang^{1,2}, Qimin Xu^{1,2*}, Cailian Chen^{1,2}

¹School of Automation and Intelligent Sensing, Shanghai Jiao Tong University

²Key Laboratory of System Control and Information Processing, Ministry of Education of China

³School of Information Science and Technology, Harbin Institute of Technology

*Email: qiminxu@sjtu.edu.cn

Abstract—Logic synthesis (LS) is a core stage in digital integrated circuit design, typically performed by applying optimized operators in Electronic Design Automation (EDA) tools. The quality of results (QoR) largely depends on the operator sequence. Traditional heuristic methods struggle with scalable circuit complexity, while existing learning-based approaches improve optimization but require retraining for each new circuit, limiting adaptability. To address this, we propose MetaSyn, a meta-reinforcement learning framework with multimodal circuit representation for adaptive logic synthesis. MetaSyn achieves both adaptability and high performance via three innovations: (1) A Model-Agnostic Meta-Learning (MAML) framework tailored for logic synthesis, where the inner and outer loops enable learning initialization parameters that allow rapid fine-tuning on unseen circuits with few samples. (2) A cooperative multi-stage RL environment (MSRL) with a multi-PPO architecture, dual-component action space, and delayed rewards, where three actor networks collaboratively optimize stage-specific operator sequences for higher performance and fast adaptability. (3) A general multimodal circuit representation (MCR) that fuses features from pre-trained DeepGate2 (AIGs), pre-trained Mamba (operator sequences), and an MLP (scalar states) via cross-attention and residual gating, forming a unified input for the policy network to enhance performance and generalization. Evaluations on the EPFL benchmark show that MetaSyn improves performance by up to 31.2% over compress2rs and by 20.8% over the state-of-the-art (SOTA) method, while offering significant advantages in fast adaptation to diverse circuits.

I. INTRODUCTION

Logic synthesis (LS) transforms RTL hardware descriptions into optimized gate-level netlists under constraints like timing, area, and power. The open-source LS tool ABC [1] provides a variety of operators for AIG optimization and mapping, and the final Quality of Results (QoR) heavily depends on the sequence of these operator executions. However, growing circuit complexity leads to an exponential expansion of the operator search space. Due to variability across different circuits, identical operations may yield different outcomes, which requires re-exploring operator sequences for each new design. To tackle these challenges and reduce reliance on heuristics, many learning-based LS methods have been proposed. DRiLLS [2] frames synthesis as a Markov Decision Process (MDP) and uses A2C for operator selection, showing promising results. BSBO [3] proposes an enhanced Bayesian optimization-based framework, while EasySO [4] introduces a hybrid discrete-continuous action space to jointly optimize operator selection and parameter tuning.

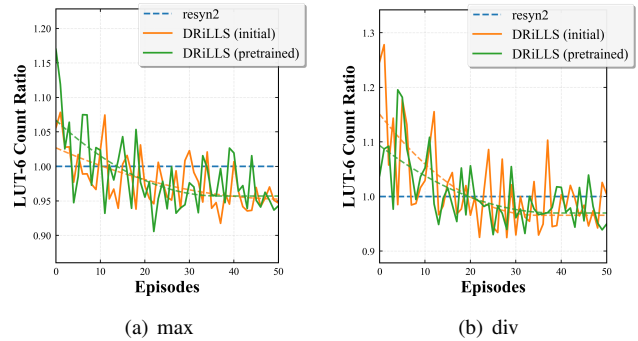


Fig. 1: Optimization curves of DRiLLS [2] on two circuits. The curves record the ratio between the minimum LUT-6 count per episode and the result of resyn2 [1], where “initial” represents training from scratch, and “pretrained” represents pretraining DRiLLS first on circuits with a similar number of nodes.

While the above methods have achieved notable improvements in final optimization quality, they lack adaptability to unseen circuits and require retraining for each new design—leading to inefficiency. As shown in Fig. 1, the performance of DRiLLS when trained from scratch and when pretrained is almost identical on the same circuit. AlphaSyn [5] and CBTune [6] improve sample efficiency by utilizing domain-specific Monte Carlo Tree Search (MCTS) and a contextual bandit (Syn-LinUCB). The latest method PIRLLS [7] combines imitation learning on expert trajectories with RL fine-tuning, enabling offline knowledge transfer and reducing runtime, while improving generalization. However, these methods often trade solution quality for efficiency and fall short in fast adaptation to unseen circuits. For instance, PIRLLS relies heavily on expert imitation, producing oversimplified sequences compared to adaptive online methods. Its generalization is also limited to the circuits with similar optimization sequences and may degrade on functionally different designs.

While online learning methods enable circuit-customized optimization, they fundamentally lack fast adaptation capability to new designs. This limitation stems from their exclusive focus on per-circuit optimization without the ability of learning to learn—failing to develop generalizable strategies for logic synthesis. It is essential to enable the agent to learn how to perform logic synthesis—much like how humans master

methodologies—such that it can generalize and adapt swiftly to new analogous tasks. Therefore, this paper proposes MetaSyn, a meta-reinforcement learning framework with multi-modal circuit representation for logic synthesis optimization, as shown in Fig. 2. Unlike existing RL agents that relearn from scratch for each circuit, MetaSyn learns like a designer who masters the methodology and adapts quickly to new designs. Our contributions are as follows:

- **MAML framework for logic synthesis.** We are the first to adapt Model-Agnostic Meta-Learning (MAML) to logic synthesis, leveraging meta-RL to learn foundational optimization policies from a diverse set of circuit tasks. Instead of training from scratch for each circuit, the model learns shared meta-parameters that enable fast, few-shot adaptation to new circuits.
- **RL environment for multi-stage optimization (MSRL).** To coordinate distinct LS stages, we introduce MSRL, featuring a multi-PPO architecture with a dual-component action space and delayed rewards. Three actor networks target key LS stages—logic optimization (LO), technology mapping (TM), and post-mapping optimization (PO)—allowing the policy to optimize cooperatively across stages and improve overall performance, which raises the optimization ceiling and ensures efficient adaptation.
- **Multimodal circuit representation (MCR) module.** To provide a holistic and generalizable state for the agent, we design the MCR module. Structural and functional features are extracted from AIGs via pre-trained DeepGate2, operator sequences via Mamba, and scalar features via MLP. These features are fused through cross-attention and residual gating into a unified representation, offering a comprehensive circuit state while the pre-training strategy enhances generalization across circuits.

II. RELATED WORK

A. RL for Logic Synthesis

Reinforcement learning has emerged as a powerful tool for automating logic synthesis, particularly in exploring optimization sequences. Hosny et al. [2] pioneered deep reinforcement learning (DRL) for logic optimization, outperforming human-crafted heuristics. Zhu et al. [8] combined graph convolutional networks with RL to learn structure-aware local transformations. A hybrid discrete-continuous action space is proposed in EasySO [4] to simultaneously optimize operator selection and parameter tuning. Recent works [9], [10], [11] have also leveraged RL for logic synthesis.

B. Meta-Learning

Meta-learning empowers models to quickly adapt to new tasks with limited data. While foundational methods like MAML [12] and Reptile [13] established this field, they struggle with modeling task structure and generalization. R2D2 [14] enhances gradient stability through task-specific embeddings and initialization. BOIL [15] emphasizes the role of representation learning over fast adaptation. More recently, Wu et al.

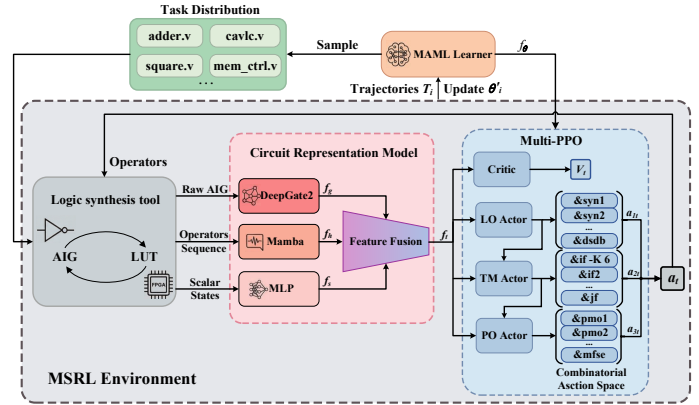


Fig. 2: The overall architecture of our MetaSyn framework.

introduced ConML [16], a unified meta-learning framework that leverages task-level contrastive learning to improve both optimization-based and metric-based approaches.

C. Circuit Representation Learning

Circuit representation learning encodes structural [17], [18], [19], functional [20], [21] information into vectors for tasks like logic synthesis and placement prediction. The DeepGate series [20], [22], [23] effectively enhances the general representation of circuits through a carefully designed graph neural network architecture and pre-training strategy. Yang et al. [24] model circuits as heterogeneous graphs incorporating both logic netlists and physical layouts, achieving top-tier performance across various benchmarks. Luo et al. [25] propose direction-equivariant hypergraph neural networks to capture long-range dependencies and directional connections.

III. PROPOSED METHOD

A. MAML Framework tailored for logic synthesis tasks

Logic synthesis requires optimizing circuits that vary widely in size, topology, and design constraints, making it difficult for conventional RL agents to generalize across tasks. This motivates the use of meta-learning: by learning transferable strategies from diverse tasks, Meta-RL facilitates fast adaptation to new circuits. We apply MAML [12], which learns a good initialization that enables quick adaptation to new synthesis tasks via a few gradient updates. MAML is a model-agnostic algorithm, which is applicable to all models whose parameters are updated through gradient descent, such as the Actor-Critic network in Proximal Policy Optimization (PPO) algorithm. In our framework, each task represents a synthesis problem for a specific circuit, sampled from a diverse training distribution.

The details of the MAML algorithm used for logic synthesis are shown in Algorithm 1. MAML employs a bilevel optimization structure: the inner loop performs fast adaptation on sampled tasks using a small support set, while the outer loop updates meta-parameters based on task-specific gradients.

$$\mathcal{L}_{\mathcal{T}_i}(f_\theta) = \mathbb{E}_{\tau \sim \mathcal{T}_i} \left[\sum_{t=1}^T \ell(s_t, a_t) \right] \quad (1)$$

Where f_θ denotes the current policy model, τ represents the trajectory obtained through interaction with the environment, and ℓ signifies the single-step loss function. Here s_t denotes the state of the logic synthesis environment at time step t and a_t refers to the action taken by the agent at time step t . More terms related to RL will be presented in the next subsection. Subsequently, the model parameters undergo one or several steps of gradient descent as follows:

$$\theta'_i = \theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta) \quad (2)$$

The adapted parameters θ'_i for the circuit task \mathcal{T}_i are thus obtained. The outer loop constitutes the meta-optimization phase, which updates the meta-parameters to enhance generalization capability. Following task-specific fast adaptation, the model’s performance on task \mathcal{T}_i is evaluated using the query set. The resulting loss is then leveraged to update the global meta-parameters θ . The overarching meta-objective minimizes the sum of post-adaptation losses across multiple tasks:

$$\min_\theta \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta'_i}) = \sum_{\mathcal{T}_i \sim p(\mathcal{T})} \mathcal{L}_{\mathcal{T}_i}(f_{\theta - \alpha \nabla_\theta \mathcal{L}_{\mathcal{T}_i}(f_\theta)}) \quad (3)$$

Through repeated updates across diverse tasks, MAML learns a meta-parameter initialization θ that is highly adaptable to new circuits. Rather than optimizing for average performance, MAML focuses on an initialization that quickly responds to circuit-specific gradients, enabling rapid fine-tuning with only a few steps. This capability is particularly valuable in logic synthesis, where efficient adaptation can reduce exploration time and accelerate convergence. Compared with prior DRL-based approaches, MAML-based RL achieves few-shot generalization without the need for extensive retraining or massive sampling, making it well-suited for the heterogeneous and large-scale nature of modern circuit optimization.

B. RL environment for cooperative multi-stage optimization

We employ an RL environment to automate the search for logic synthesis strategies. Existing RL-based methods primarily focus on LO alone, ignoring TM and PO. However, choices made in LO directly affect TM and PO performance, and isolated optimization often results in suboptimal QoR. To capture these dependencies, we design MSRL to optimize all stages cooperatively. Unlike other previous works, our RL environment cooperatively optimizes multiple stages of logic synthesis, which facilitates faster exploration of optimal results, while simultaneously enhancing both performance and efficiency.

Multi-PPO architecture. To bridge the optimization gap across stages, we adopt a multi-PPO framework. Logic synthesis is modeled as an MDP, where the agent selects operations a_t based on the current state s_t and receives reward r_t . Since LO aims to reduce nodes while TM and PO target LUT count and other metrics, their objectives differ and cannot be optimized independently. Our multi-PPO design unifies stage-specific optimizations within a shared environment, while

Algorithm 1 MAML for Logic Synthesis Optimization

Require: $p(\mathcal{T})$: distribution over circuit synthesis tasks

Require: α, β : step size hyperparameters

```

1: Randomly initialize policy parameters  $\theta$ 
2: while not done do
3:   Sample batch of circuit tasks  $T_i \sim p(\mathcal{T})$ 
4:   for all task  $T_i$  do
5:     Interact with circuit environment of  $T_i$  to collect  $K$ 
       support trajectories
6:      $D = \{(s_1, a_1), \dots, (s_H, a_H)\}$  using  $f_\theta$ 
7:     Compute task-specific loss on support set:
8:      $\mathcal{L}_{T_i}(f_\theta) = \mathbb{E}_{\tau \sim p} [\sum_t \ell(s_t, a_t)]$ 
9:     Compute adapted parameters:
10:     $\theta'_i = \theta - \alpha \nabla_\theta \mathcal{L}_{T_i}(f_\theta)$ 
11:    Interact again using  $f_{\theta'_i}$  to collect query trajectories
12:     $D'_i = \{(s_1, a_1), \dots, (s_H, a_H)\}$ 
13:  end for
14:  Update meta-parameters:
15:   $\theta \leftarrow \theta - \beta \nabla_\theta \sum_{T_i \sim p(\mathcal{T})} \mathcal{L}_{T_i}(f_{\theta'_i})$ 
16: end while

```

PPO’s clipped objective constrains policy divergence and ensures stable updates.

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t \left[\min \left(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t \right) \right] \quad (4)$$

where $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ denotes the probability ratio between new and old policies, \hat{A}_t represents the advantage function (we use GAE), and ϵ is the hyperparameter controlling the clipping range. As shown in Fig. 2, each actor receives the shared representation and makes decisions specific to its synthesis stage. The TM actor and PO actor additionally receive action information from the previous actor. All actors are initialized from a MAML-based meta-policy, allowing fast adaptation via few gradient updates. This architecture enables coordinated multi-stage optimization under a unified objective. The agent observes a state vector s_t that integrates multi-source information: circuit scalar states, AIG features extracted by a pre-trained DeepGate2 model, and historical operator sequences features encoded by a pre-trained Mamba model. The scalar features include the following:

- Number of nodes, edges, and levels of the circuit.
- NOT gate percent and AND gate percent.
- LUT-6 count and levels after technology mapping.
- The length of the current operator sequence.

Details about other representations are described in the section III-C.

Dual-component action space. To mitigate the exponential growth in action space dimensionality from cooperative multi-stage optimization that increases search difficulty and reduces learning efficiency, we redesign the action space using domain knowledge. Instead of relying solely on basic operators, we introduce a composite space with two components: basic operators and effective scripts. Leveraging ABC9 commands, we incorporate manual scripts (e.g., $\text{syn1} = \{rw, rf, b\}$) alongside retained some effective basic operators (e.g., $\&dsdb$), significantly compressing the action space while boosting optimization capability.

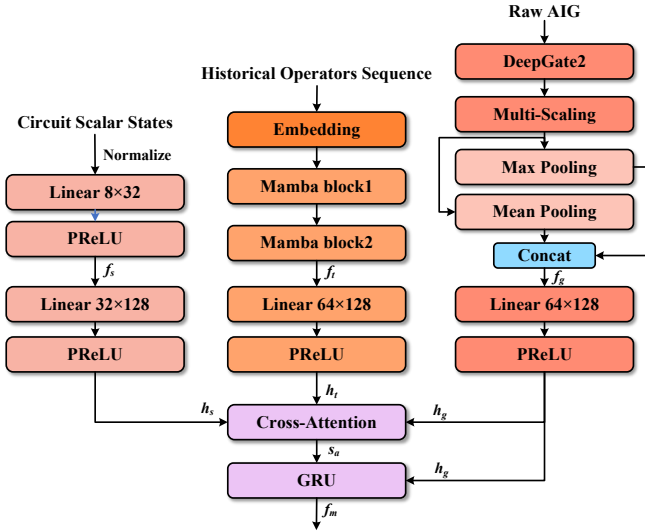


Fig. 3: The architecture of our MCR module.

Delayed reward function. Reward design in logic synthesis is difficult due to multi-objective trade-offs, stage differentiation, and weak action-result causality. Prior RL methods often adopt dense step-wise rewards, which in long sequences bias agents toward early local optima, undervalue later operations, and deviate from global QoR. To overcome this, we design a delayed reward with a large global reward only at the sequence end:

$$r_t = \begin{cases} \alpha \cdot \frac{L_{\text{ini}} - L_{\text{min}}}{L_{\text{ini}}}, & \text{if } t = \ell_{\text{seq}} \\ \frac{L_{\text{min}}(s_t) - L(s_{t+1})}{L_{\text{min}}(s_t)}, & \text{if } t < \ell_{\text{seq}} \\ & \text{and } L(s_{t+1}) < L_{\text{min}}(s_t) \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

where L_{min} is the episode’s global minimum LUT-6 count, α is a scaling factor (set to 100), and ℓ_{seq} is the max sequence length. $L_{\text{min}}(s_t)$ is the minimum LUT-6 count up to step t . This scheme provides small incremental rewards for improvements and a large terminal reward, aligning optimization with global QoR, improving sample efficiency, and generalizing across different circuits.

C. Multimodal circuit representation module

Existing methods rely on hand-crafted or single-modal features, limiting generalization across heterogeneous circuits. To overcome this, we seek to develop an efficient circuit representation with strong generalization capabilities—one that can effectively boost the agent’s performance and adaptability. We propose a multimodal circuit representation (MCR) module that integrates structural, sequential, and scalar information into a unified embedding. Its architecture is shown in Fig. 3.

We utilize the pre-trained DeepGate2 [22] model to extract structural and functional features from the AIGs. Through function-aware supervision and a one-round GNN architecture, DeepGate2 can effectively capture the structural and functional information of circuits, producing two (K, D) high-dimensional vector (for structure and function). To reduce

the dimensionality, we concatenated these two vectors and then performed multi-scale pooling. For pre-training, we use the open-source dataset from Shi et al. [22], which includes 10,824 circuits from ITC’99 [26], IWLS’05 [27], EPFL [28], and Open-Core [29] benchmarks. To reduce computational overhead during synthesis, we reuse extracted features if the AIG node count changes by less than 20% between steps; otherwise, DeepGate2 is reinvoked.

To capture temporal dependencies in operator sequences, we use a pre-trained Mamba encoder—a state-space model efficient for long-range modeling. We pre-train the Mamba encoder on 10,000 operator sequences collected during debugging. The training involves two objectives: masked operator prediction, where 30% of tokens are randomly masked and reconstructed to capture local context; and contrastive learning, where subsequences from the same circuit form positive pairs, and those from different circuits form negatives. These two pretraining tasks enhance its ability to encode sequences and generalize across circuits.

To efficiently fuse these features, we design a hierarchical cross-modal fusion architecture to jointly represent the circuit optimization state. Each modality is first independently encoded, and all three vectors are projected into a shared 128-D space and normalized. We then apply a graph-guided cross-attention mechanism for selective fusion, using \mathbf{h}_g as the query and $[\mathbf{h}_s; \mathbf{h}_t]$ as keys and values. The fusion features based on attention weights \mathbf{s}_a is calculated as follows

$$\mathbf{s}_a = \text{softmax} \left(\frac{(\mathbf{W}_q \mathbf{h}_g)(\mathbf{W}_k [\mathbf{h}_s; \mathbf{h}_t])^\top}{\sqrt{d_k}} \right) \mathbf{W}_v [\mathbf{h}_s; \mathbf{h}_t] \quad (6)$$

where \mathbf{W}_q , \mathbf{W}_k and \mathbf{W}_v are the weight matrix of queries, keys, and values. d_k refers to dimension of Key vectors. To balance fused and original information, a residual gate is learned as:

$$\mathbf{g} = \sigma(\mathbf{W}_g \mathbf{h}_g + \mathbf{W}_f \mathbf{s}_a) \quad (7)$$

where \mathbf{W}_g and \mathbf{W}_f are both learnable matrix. The final unified state representation f_m is computed by element-wise mixing:

$$\mathbf{f}_m = \mathbf{g} \odot \mathbf{s}_a + (1 - \mathbf{g}) \odot \mathbf{h}_g \quad (8)$$

This architecture can identify important features while preserving original information, and efficiently fuse multi-modal circuit state representations. The 128-D vector f_m is directly input to the policy network as the circuit state observation.

In summary, MetaSyn combines meta-learning, cooperative multi-stage optimization, and multimodal representation into a unified framework. MAML ensures fast few-shot adaptability, MSRL raises the global optimization ceiling by coordinating multiple stages, and MCR provides a robust representation foundation for policy learning. Together, these components address the key challenges of efficiency, generalization, and performance in logic synthesis.

IV. EXPERIMENTS

We conducted comprehensive experiments to evaluate the proposed MetaSyn framework. Next, we will elaborate on the details of our experiments and present the experimental results.

TABLE I: Comparison results with high-performance methods.

Benchmark	compress2rs		DRiLLS [2]			BSBO [3]			EasySO [4]			MetaSyn		
	LUTs	Levels	LUTs	Impr.(%)	Levels	LUTs	Impr.(%)	Levels	LUTs	Impr.(%)	Levels	LUTs	Impr.(%)	Levels
adder	243	51	243.3	-0.1%	51	229	5.8%	51	229	5.8%	51	191.2	21.3%	53.9
cavlc	122	4	111.8	8.4%	4	100	18.0%	4	94	23.0%	4	94.6	22.5%	4.2
ctrl	31	2	28.0	9.7%	2	26	16.1%	2	26	16.1%	2	25	19.4%	2
div	5219	863	7564.2	-44.9%	824	4675	10.4%	830	4534	13.1%	850	3426.1	34.3%	852.5
i2c	290	4	298.5	-2.9%	4	242	16.6%	4	232	20.0%	3	234.1	19.3%	4
int2float	47	3	43.6	7.2%	3	35	25.5%	3	33	29.8%	3	31	34.0%	3.8
log2	7995	72	7643.8	4.4%	72	8702	-8.8%	71	7131	10.8%	67	6628.5	17.1%	78.7
max	770	42	732.4	4.9%	36	741	3.8%	36	722	6.2%	41	647.8	15.9%	43.5
mem_ctrl	11197	25	10894.9	2.7%	19	3195	71.5%	22	3217	71.3%	25	4262.3	61.9%	18.9
multiplier	5958	53	5698.5	4.4%	48	5538	7.0%	52	5085	14.6%	53	4818.3	19.1%	56.4
priority	160	22	138.3	13.6%	15	128	20.0%	16	117	26.9%	16	109.2	31.8%	23.6
router	56	5	70.1	-25.2%	4	27	51.8%	5	27	51.8%	4	27	51.8%	4.9
sin	1460	36	1439.1	1.4%	33	1499	-2.7%	36	1294	11.4%	36	1225.3	16.1%	38.7
sqrt	3351	1031	4591.4	-37.0%	1005	3329	0.7%	1005	3169	5.4%	994	3006.1	10.3%	1009.1
square	3919	50	3893.7	0.6%	49	3375	13.9%	37	3326	15.1%	50	3209.6	18.1%	50.4
voter	1744	13	1783.8	-2.3%	12	1355	22.3%	13	1345	22.9%	12	1327.5	23.9%	12.2
average	2660.1	142.3	2823.5	-6.1%	136.3	2074.8	22.0%	136.7	1911.3	28.1%	138.2	1829.0	31.2%	143.6

A. Implementation Details

Our method is implemented on Python v3.12.9 using PyTorch v2.6 and the MSRL environment is implemented by developing python interfaces for ABC [1]. We conduct the experiments on a 24-cores 3.8 GHz AMD Ryzen Threadripper Pro 5965WX CPU and an NVIDIA RTX 3090 GPU, CUDA 12.6.

The evaluation metric for the performance of the method is the minimum LUT-6 count that meets the levels constraint (not exceeding 10% of the *compress2rs* script in ABC [1]). We also compare efficiency with other methods. Since running time is greatly affected by hardware configurations and optimization strategies, and a time cost that does not grow exponentially is acceptable for logic synthesis tasks. So it is unreasonable to blindly reduce optimization time. We introduce the balance factor F_b to evaluate the trade-off between performance and efficiency among different methods, whose calculation method is as follows.

$$F_b = \beta \frac{\delta}{\ln(\tau)} \quad (9)$$

Where δ represents the average optimization improvement rate of the method relative to *compress2rs*, and τ stands for the average running time. β is a scaling factor (set to 10). F_b can well reflect the trade-off between performance and efficiency among different methods. MetaSyn run with 10 different random seeds, and the experimental results are averaged.

B. Performance Evaluation

We first conducted a performance evaluation on 16 EPFL benchmarks, using the *compress2rs* script as the baseline, and compared the optimal results of our MetaSyn with those of DRiLLS [2], BSBO [3] and EasySO [4].

We optimized each circuit for 50 episodes with a maximum operator sequence length of 25. The results obtained are shown in Table I, where "LUTs" represents the minimum LUT-6 count obtained during the optimization process, and "Impr."

denotes the improvement rate relative to *compress2rs*. Compared with *compress2rs*, our MetaSyn reduces the LUT count by 31.2% while not exceeding 10% of the *compress2rs*, and it also achieves a 28.6% reduction compared to DRiLLS, the earliest DRL method for logic synthesis. Compared with BSBO and EasySO, our method achieves performance improvements of 16.5% and 9.4% respectively. This is sufficient to prove that our method has a high performance and optimization upper limit. Notably, on large-scale circuits such as log2 and div, MetaSyn achieves up to 17.1% and 34.3% improvements, respectively. This demonstrates the benefit of coordinating multiple optimization stages: by jointly optimizing LO, TM, and PO, the model avoids the suboptimal local trade-offs common in single-stage methods.

C. Fast adaptability verification

To demonstrate the fast adaptation capability of our MetaSyn to new circuits, we use EPFL benchmark suite as the task distribution. MetaSyn is trained by sampling from EPFL benchmarks to learn a meta-policy, after which few-shot adaptation to each circuit are performed on this basis. We compare the optimization performance and running time with AlphaSyn [5], CBTune [6], PIRLLS [7]. The optimization results of MetaSyn all meet the levels constraints. As shown in Table II, the average running time of MetaSyn with few-shot adaptation is about 30% longer than that of PIRLLS, while its LUTs are reduced by 22.1%, 19.6%, and 15.3% compared with those of AlphaSyn, CBTune, and PIRLLS respectively. Moreover, MetaSyn also has significant advantages in terms of running speed on small-scale circuits such as adder and ctrl, attributed to the strong initialization from MAML and lightweight pre-trained representations in MCR. By comparing the F_b (equation 9) of different methods, it can be concluded that our MetaSyn achieves the best trade-off between performance and efficiency, with its F_b being 120.2% higher than that of the SOTA method PIRLLS.

In addition, to demonstrate that our method is capable of rapidly adapting to completely unseen new circuits, we applied the meta-optimizer derived from the EPFL benchmark suite

TABLE II: Comparison results with high-efficiency methods.

Benchmark	compress2rs	AlphaSyn [5]			CBTune [6]			PIRLLS [7]			MetaSyn (few-shot)		
	LUTs	LUTs	Impr.(%)	rt(m)	LUTs	Impr.(%)	rt(m)	LUTs	Impr.(%)	rt(m)	LUTs	Impr.(%)	rt(m)
adder	243	244	-0.4%	6.14	244	-0.4%	5.97	196.7	19.1%	1.62	195.5	19.6%	0.49
cavlc	122	106	13.1%	5.35	111	9.0%	2.37	107.6	11.8%	1.64	102.2	16.2%	0.21
ctrl	31	28	9.7%	5.68	28	9.7%	0.59	27.0	12.9%	1.66	25.6	17.4%	0.18
div	5219	6650.1	-27.4%	11.88	4180.91	19.9%	25.58	3963.3	24.1%	17.52	3931.5	24.7%	17.26
i2c	290	280	3.5%	6.22	283.11	2.4%	3.61	271.3	6.5%	1.85	245.6	15.3%	0.47
int2float	47	39	17.0%	5.54	40	14.9%	2.76	39.0	17.0%	1.54	34.3	27.0%	0.32
log2	7995	7580	5.2%	11.78	7580	5.2%	41.27	7469.6	6.6%	12.80	7079.4	11.5%	21.72
max	770	680	11.7%	5.70	684.25	11.1%	6.01	675.6	12.3%	2.49	667.6	13.3%	2.53
mem_ctrl	11197	9513.2	15.0%	10.99	10242.57	8.5%	45.81	9547.5	14.7%	17.39	5672.3	49.3%	28.53
multiplier	5958	5672	4.8%	10.34	5679.75	4.7%	29.08	5663.6	4.9%	10.57	5432.9	8.8%	22.62
priority	160	135	15.6%	5.83	138.86	13.2%	3.41	126.3	21.1%	1.58	115.4	27.9%	0.38
router	56	65	-16.1%	5.34	68.11	-21.6%	2.32	61.3	-9.5%	1.64	29.5	47.3%	0.67
sin	1460	1438	1.5%	6.77	1441.67	1.3%	9.71	1434.75	1.7%	3.57	1297.1	11.2%	21.25
sqrt	3351	4415	-31.8%	9.83	4607	-37.5%	36.51	3837	-14.5%	17.81	3277.6	2.2%	6.27
square	3919	3877	1.1%	8.72	3882.11	0.9%	25.99	3849.5	1.8%	7.92	3504.6	10.6%	11.71
voter	1744	1537.4	11.9%	7.84	1682.25	3.5%	11.46	1579	9.5%	3.70	1379.5	20.9%	9.32
average	2660.1	2641.2	0.7%	7.75	2555.8	3.9%	15.78	2428.1	8.7%	6.58	2061.9	22.5%	9.00
F_b	-	-	0.035	-	-	0.142	-	0.463	-	-	1.024	-	-

TABLE III: The results of different initialization strategies on 5 circuits of ISCAS85

Benchmark	compress2rs	random	pre-trained	MetaSyn
c2670	101	85.6	83.3	81.2
c3540	244	211.3	224.9	202.4
c5315	268	235	234.7	229.1
c7552	326	277.8	286	264
c6288	521	352.8	375.1	346.5
average	292.0	243.1	240.7	224.4

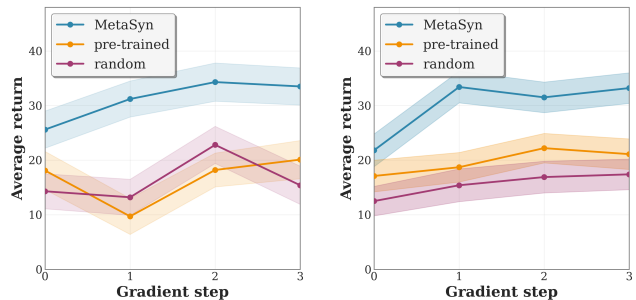
TABLE IV: Results of ablation study

Module	w/o MAML	w/ MAML
RL-PPO	-1.8%	0.3%
MSRL	10.1%	16.9%
MSRL+Transformer	11.2%	17.8%
MSRL+Mamba	12.1%	18.9%
MSRL+MCR	17.2%	24.6%

to 5 circuits selected from ISCAS85 for few-shot adaptation. We compared the results with random initialization and a pre-training policy under the same number of samples. It should be noted that all methods were conducted in our MSRL environment. As shown in Table III, MetaSyn clearly achieved the best results with the meta-policy. We also plotted the average return curves of different initialization strategies over four episodes optimization on the c7552 and c6288 circuits, as shown in Fig. 4. It is evident that MetaSyn can find the optimization direction faster and achieve better results. Notably, the pretraining policy is sometimes even inferior to random initialization. In conclusion, our MetaSyn can fast adapt to and optimize unseen circuits.

D. Ablation study

We conducted ablation study to demonstrate the effectiveness of each component in MetaSyn. We compare the final average LUT-6 count reduction rates (relative to *compress2rs*) across 16 EPFL benchmarks under different setting. The results are shown in Table IV, where "w/o MAML" indicates training directly on all circuits without meta-learning; "w/ MAML" indicates performing meta-learning first under the



(a) c7552

(b) c6288

Fig. 4: Comparison of different initialization strategies

MAML framework and then adapting on all circuits. All circuits are optimized for 10 episodes. "RL-PPO" refers to using the conventional PPO architecture with the action space of DRiLLS [2]. It can be considered that achieving higher performance within a same small number of episodes under the same method complexity indicates that the model has faster adaptation capability. The results show that MSRL significantly improves performance, MAML prominently enhances the model's fast adaptation capability, and MCR further enhances the model's performance and generalization ability.

V. CONCLUSION

In this paper, we introduced MetaSyn, the first meta-reinforcement learning framework for adaptive logic synthesis. By integrating a MAML-based initialization, a cooperative multi-stage RL environment, and a multimodal circuit representation, MetaSyn achieves both high performance and rapid few-shot adaptability across diverse circuits. Extensive experiments on the EPFL and ISCAS85 benchmarks demonstrate that MetaSyn not only surpasses state-of-the-art methods in optimization quality but also generalizes effectively to unseen circuits with minimal fine-tuning. Beyond improving logic synthesis, this work demonstrates the potential of meta-RL in EDA, suggesting its applicability for more adaptive design strategies in large-scale or heterogeneous circuits.

REFERENCES

- [1] R. Brayton and A. Mishchenko, "Abc: An academic industrial-strength verification tool," in *Computer Aided Verification: 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings* 22. Springer, 2010, pp. 24–40.
- [2] A. Hosny, S. Hashemi, M. Shalan, and S. Reda, "Drills: Deep reinforcement learning for logic synthesis," in *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2020, pp. 581–586.
- [3] C. Feng, W. Lyu, Z. Chen, J. Ye, M. Yuan, and J. Hao, "Batch sequential black-box optimization with embedding alignment cells for logic synthesis," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–9.
- [4] J. Yuan, P. Wang, J. Ye, M. Yuan, J. Hao, and J. Yan, "Easysso: Exploration-enhanced reinforcement learning for logic synthesis sequence optimization and a comprehensive rl environment," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.
- [5] Z. Pei, F. Liu, Z. He, G. Chen, H. Zheng, K. Zhu, and B. Yu, "Alphasyn: Logic synthesis optimization with efficient monte carlo tree search," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 2023, pp. 1–9.
- [6] F. Liu, Z. Pei, Z. Yu, H. Zheng, Z. He, T. Chen, and B. Yu, "Cbtune: Contextual bandit tuning for logic synthesis," in *2024 Design, Automation Test in Europe Conference Exhibition (DATE)*, 2024, pp. 1–6.
- [7] G. Dong, J. Zhai, H. Cheng, X. Yang, C. Shi, and K. Zhao, "Pirlls: Pretraining with imitation and rl finetuning for logic synthesis," in *Proceedings of the 30th Asia and South Pacific Design Automation Conference*, 2025, pp. 65–71.
- [8] K. Zhu, M. Liu, H. Chen, Z. Zhao, and D. Z. Pan, "Exploring logic optimizations with reinforcement learning and graph convolutional network," in *Proceedings of the 2020 ACM/IEEE Workshop on Machine Learning for CAD*, 2020, pp. 145–150.
- [9] A. F. Budak, Z. Jiang, K. Zhu, A. Mirhoseini, A. Goldie, and D. Z. Pan, "Reinforcement learning for electronic design automation: Case studies and perspectives," in *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2022, pp. 500–505.
- [10] G. Zhou and J. H. Anderson, "Area-driven fpga logic synthesis using reinforcement learning," in *Proceedings of the 28th Asia and South Pacific Design Automation Conference*, 2023, pp. 159–165.
- [11] G. Pasandi, S. Pratty, and J. Forsyth, "Aisyn: Ai-driven reinforcement learning-based logic synthesis framework," *arXiv preprint arXiv:2302.06415*, 2023.
- [12] C. Finn, P. Abbeel, and S. Levine, "Model-agnostic meta-learning for fast adaptation of deep networks," in *International conference on machine learning*. PMLR, 2017, pp. 1126–1135.
- [13] A. Nichol and J. Schulman, "Reptile: a scalable metalearning algorithm," *arXiv preprint arXiv:1803.02999*, vol. 2, no. 3, p. 4, 2018.
- [14] L. Bertinetto, J. F. Henriques, P. H. Torr, and A. Vedaldi, "Meta-learning with differentiable closed-form solvers," *arXiv preprint arXiv:1805.08136*, 2018.
- [15] J. Oh, H. Yoo, C. Kim, and S.-Y. Yun, "Boil: Towards representation change for few-shot learning," *arXiv preprint arXiv:2008.08882*, 2020.
- [16] S. Wu, Y. Wang, Y. Bian, and Q. Yao, "Conml: A universal meta-learning framework with task-level contrastive learning," *arXiv preprint arXiv:2410.05975*, 2024.
- [17] K. Zhu, H. Chen, W. J. Turner, G. F. Kokai, P.-H. Wei, D. Z. Pan, and H. Ren, "Tag: Learning circuit spatial embedding from layouts," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–9.
- [18] A. Fayyazi, S. Shababi, P. Nuzzo, S. Nazarian, and M. Pedram, "Deep learning-based circuit recognition using sparse mapping and level-dependent decaying sum circuit representations," in *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2019, pp. 638–641.
- [19] Z. He, Z. Wang, C. Bai, H. Yang, and B. Yu, "Graph learning-based arithmetic block identification," in *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*. IEEE, 2021, pp. 1–8.
- [20] M. Li, S. Khan, Z. Shi, N. Wang, H. Yu, and Q. Xu, "Deepgate: Learning neural representations of logic gates," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 667–672.
- [21] Z. Wang, C. Bai, Z. He, G. Zhang, Q. Xu, T.-Y. Ho, B. Yu, and Y. Huang, "Functionality matters in netlist representation learning," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 61–66.
- [22] Z. Shi, H. Pan, S. Khan, M. Li, Y. Liu, J. Huang, H.-L. Zhen, M. Yuan, Z. Chu, and Q. Xu, "Deepgate2: Functionality-aware circuit representation learning," in *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*. IEEE, 2023, pp. 1–9.
- [23] Z. Shi, Z. Zheng, S. Khan, J. Zhong, M. Li, and Q. Xu, "Deepgate3: Towards scalable circuit representation learning," in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, 2024, pp. 1–9.
- [24] S. Yang, Z. Yang, D. Li, Y. Zhang, Z. Zhang, G. Song, and J. Hao, "Versatile multi-stage graph neural network for circuit representation," *Advances in Neural Information Processing Systems*, vol. 35, pp. 20 313–20 324, 2022.
- [25] Z. Luo, T. S. Hy, P. Tabaghi, M. Defferrard, E. Rezaei, R. M. Carey, R. Davis, R. Jain, and Y. Wang, "De-hnn: An effective neural model for circuit netlist representation," in *International Conference on Artificial Intelligence and Statistics*. PMLR, 2024, pp. 4258–4266.
- [26] S. Davidson, "Characteristics of the itc'99 benchmark circuits," in *IEEE International Test Synthesis Workshop (ITSW)*, 1999, p. 87.
- [27] C. Albrecht, "Iwls 2005 benchmarks," in *International Workshop for Logic Synthesis (IWLS)*, vol. 9, 2005.
- [28] L. Amarú, P.-E. Gaillardon, and G. De Micheli, "The eplf combinational benchmark suite," in *Proceedings of the 24th International Workshop on Logic & Synthesis (IWLS)*, 2015.
- [29] O. Team, "Opencores," in <https://opencores.org/>.