Learning to Approximate: Circuit Learning and Deep Reinforcement Learning for Approximate Logic Synthesis with an Error Rate Guarantee

Chi-Wei Chen[1], Yi-Ting Li[1], Wuqian Tang[1], Yung-Chih Chen[2,3], Jian-Meng Yang[3], and Chun-Yao Wang[1,3]

[1]National Tsing Hua University, Taiwan, ROC
[2]National Taiwan University of Science and Technology, Taiwan, ROC
[3]ARCULUS SYSTEM CO., LTD., US

***Abstract*—Approximate computing is an emerging design paradigm for error-tolerant applications, such as multimedia processing and neural network acceleration, which enables significant reductions in circuit area, delay, or power consumption through controlled accuracy trade-offs. This paper presents a novel deep reinforcement learning (DRL)-based framework for approximate logic synthesis (ALS) augmented with a backtracking mechanism, aimed at minimizing the area–delay product (ADP) while satisfying error rate constraints. The experimental results demonstrate that our approach can reduce the ADP by up to 92.83%, and 56.79% on average under a 5% error rate constraint.**

I. Introduction

With the advancement of technology and the rapid growth of design complexity, energy efficiency has become a crucial concern [2]. To overcome this challenge, designers propose to minimize the area, delay, or power consumption of a circuit while sacrificing minor accuracy. Hence, approximate computing emerges as a promising energy-efficient paradigm for error-tolerant applications, such as image processing and deep neural network accelerators. Approximate computing carefully introduces some errors to a circuit to gain the benefits over other design metrics. If the errors are precisely managed, the functionality of a circuit is almost unaffected.

At the circuit level, approximate logic synthesis (ALS) aims to synthesize approximate versions of arbitrary circuits from its specification, under error constraints [6] [7] [9] [10] [11] [14] [16]. Some of the methods systematically applying local approximate changes (LACs) such as constant node replacement [6], node merging [16], or resubstitution with approximate care sets [9] [10], to transform the circuit iteratively while monitoring statistical error metrics. Recent ALS has leveraged heuristic search [6] [14], simulation-guided estimation [9] [10], evolutionary algorithms [6], or reinforcement learning (RL) [11], striking various balances between circuit quality and runtime.

This work was supported in part by the National Science and Technology Council (Taiwan) under Grant MOST 111-2221-E-007-121, Grant MOST 111-2221-E-011-137-MY3, Grant NSTC 112-2218-E-007-014, Grant NSTC 112-2221-E-007-106-MY2, Grant NSTC 112-2221-E-007-108, Grant NSTC 112-2425-H-007-002, and Grant NSTC 113-2425-H-007-004, Grant NSTC 113-2640-E-011-003, Grant NSTC 113-2221-E-007-082-MY3, Grant NSTC 114-2221-E-007-125-MY3, Grant NSTC 114-2218-E-007-002, Grant NSTC 114-2221-E-007-126-MY3, Grant NSTC 114-2425-H-007-002, and National Tsing Hua University under NTHU 113A0257EX and 114A0078EX.

Artificial Intelligence (AI), particularly RL, has emerged as an innovative approach to ALS recently, offering significant advantages over traditional heuristic methods. RL's ability to learn from interactions with the environment enables it to navigate the vast design spaces inherent in ALS, identifying optimization strategies that balance trade-offs among circuit area, delay, or error rates. One of the primary strengths of RL is its capability for intelligent exploration. Unlike deterministic algorithms that may be trapped in local optima, RL agents can discover more possibilities that lead to more efficient circuit implementations. For instance, the work *Q-ALS* [11] has demonstrated the effectiveness of RL in technology mapping by employing Q-learning to select optimal node approximations, resulting in reductions in area and delay while adhering to error rate constraints.

However, integrating RL into ALS faces several challenges. First, enabling an RL agent to make decisions requires a comprehensive understanding of the circuit's functionality and structural properties. Capturing this information in the agent's state representation is not trivial, as it needs to encompass both logical behavior and topological features to guide effective approximations. Next, directly applying LAC methods without any processing leads to a large action space. Each node in a circuit may have multiple approximations if without filtering or prioritization. The agent has a combinatorial explosion of choices. This vast action space impedes learning efficiency and may prevent the agent from converging to optimal strategies. Last, many existing ALS methods focus mainly on area reduction, often neglecting delay optimization. In practice, balancing area and delay is crucial, as improvements in one metric can adversely affect the other. Therefore, RL frameworks must incorporate multi-objective optimization to ensure that approximations do not compromise any critical performance parameters.

To address the above challenges, we propose a novel DRL-based ALS framework, which integrates the exploration capability of RL and modifies a circuit's sub-function from both global observation and local optimization. Our main contributions are as follows:

1) We propose a DRL-based framework that integrates a circuit representation learning model into the RL agent, enabling the agent to make functionality-aware decisions.

978-3-9826741-1-7/DATE26/© 2026 EDAA

2) We significantly reduce the RL action space by incorporating a structured LAC, which enhances the scalability of our approach.
3) We propose an effective error rate estimation technique that uses localized simulation values to guide LAC selection during training, which accelerates convergence without violating the error rate constraint.

II. Preliminaries

A. Error Metric

In the evaluation of approximate circuits, several error metrics are employed to quantify the quality of approximate circuits. Among these, error magnitude (EM) [12], error distance (ED) [8], and error rate (ER) [3] [4] [5] [6] [16] are commonly used. In this work, we adopt the error rate as the error metric due to its widespread use and its suitability for assessing the functional correctness of approximate circuits. The error rate is calculated as the ratio of input patterns that result in incorrect values in the primary outputs (POs). That is, error rate is calculated as EQ (1):

$$\mathrm{ER} = \frac{1}{2^I} \sum_{i=1}^{2^I} \delta\left(\hat{y}_i \neq y_i\right) \tag{1}$$

where the term y_i denotes the correct output produced by the original circuit for the i^{th} input pattern, while $\hat{y}_i$ represents the output generated by the approximate circuit for the same input pattern. $\delta(\cdot)$ is an indicator function, which equals 1 if the outputs differ, and 0 otherwise. I is the number of primary inputs in the circuit.

B. Reinforcement Learning

Reinforcement Learning (RL) is a machine learning paradigm in which an agent learns to make decisions through trial-and-error interactions with an environment. At each step, the agent observes a state, selects an action, and receives a reward that reflects the quality of the action. The environment then transits to a new state, and this loop continues over time. The agent's goal is to learn a policy for choosing actions that maximizes the cumulative reward across many steps.

This formulation is particularly suitable for problems like ALS, where decisions is made iteratively for applying LAC. By framing ALS as an RL task, we can leverage this trial-and-error process to automatically learn effective approximation strategies that satisfy error rate constraints while optimizing design metrics such as area and delay.

A common approach to training RL agents is the actor–critic framework. In this setup, the **policy network (actor)** outputs the probabilities of different actions given the current state, while the **value network (critic)** estimates the expected long-term reward from that state. The critic provides feedback to stabilize and guide the learning of the policy, reducing variance and accelerating convergence. Among modern RL algorithms with actor–critic framework, *proximal policy optimization (PPO)* has emerged as a popular choice because it balances performance and training stability. It incorporates a mechanism to limit overly large policy updates, which improves robustness without adding the heavy computational burden. This makes PPO well suited for high-dimensional or combinatorial decision problems, where both effective exploration and stability are crucial. Hence, we use the PPO in this work.

III. The Proposed ALS Framework

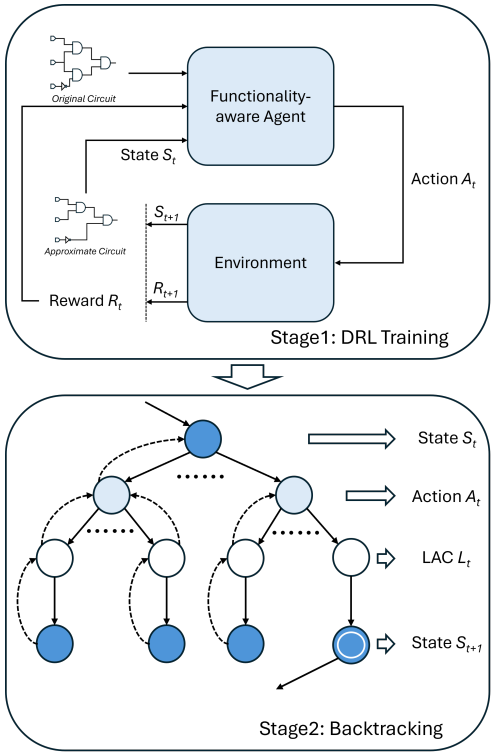

Fig. 1: The Proposed ALS Framework.

In this section, we present the proposed DRL-based ALS framework. We first introduce our overall framework in Section III-A, then we describe each part of the framework in the succeeding sections.

A. Overview of Proposed Framework

As in Fig. 1, our proposed framework consists of two main stages: DRL training and backtracking. In the first stage, a functionality-aware agent is trained to select target nodes for LACs based on graph-level circuit embeddings, then LACs are derived using deterministic heuristics and evaluated through efficient error rate estimation in the RL environment. The agent receives rewards mainly based on the area reduction ratio, guiding it to make approximations under the error rate constraint. To improve robustness, the second stage introduces a backtracking mechanism that reverts to the previous state if no feasible LACs are found, allowing the ALS to recover from dead-ends and continue optimization.

A central component of our framework is the concept of LAC, which simplifies circuits by modifying local structures. At each time step, once the RL agent selects a target node, we apply an LAC by identifying new suitable fanins from the transitive fanin (TFI) cone of the target node. Restricting fanins to the TFI ensures that functional support is preserved and prevents dependency loops. For the new fanin pair, we derive an approximate function to replace the function of the target node. This new function simplifies the local structure, thereby reducing area and delay. Each LAC is then evaluated

TABLE I: Mapping between Reinforcement Learning concepts and our ALS.

RL Concept	Our ALS
State S_t	Approximate circuit at time step t
Action A_t	Select a target node for LAC
State Transition	Apply LAC candidate(s)
Reward R_t	Area reduction ratio, exploration incentive, and penalty on constraint violation
Policy π_θ	Functionality-aware agent
Backtracking	Rollback on dead-ends: revert to previous feasible state and continue if there is no feasible LAC.

using a localized, pessimistic error rate estimation to ensure that applying it does not violate the global error rate constraint. By structuring the flow in terms of (i) target node, (ii) new fanins, and (iii) approximate function, we integrate circuit-level transformations into the RL framework, making state transitions during each episode.

Table I summarizes the one-to-one correspondence between RL concepts (state, action, reward, transition, policy) and the components of the ALS. This unified view clarifies how the proposed framework combines circuit-level heuristics with learning-based decision making.

To further improve efficiency, we incorporate an infeasible target node pruning mechanism into the LAC selection process. In particular, if a chosen target node cannot yield any LAC that reduces circuit area, the node is marked as infeasible and excluded from future selections. The agent will reselect a new target node for circuit approximation. This pruning step avoids waste of learning episodes in unproductive actions and accelerates convergence by steering the agent toward nodes with higher optimization potential.

B. Action Space

Our approach separates the selection of new fanins and the derivation of approximate function of the target node from the agent to reduce the action space and increase the capability of exploration. We deterministically select fanins and derive approximate function after the agent selects the target node, which significantly reduces the action space and increases the learning performance.

The approximate circuit at each time step t is considered as state S_t, among which the terminal states are the circuits that exceed the given error rate constraint. Given the current state S_t, the set of valid action candidates that can be selected as the action A_t is the set of active nodes in S_t. An active node in AIG is a node whose transitive fanout (TFO) cone comprises at least one PO; otherwise, the node is considered inactive. We consider only active nodes as valid actions due to their effects on the functionality of the circuit. For inactive nodes, we employ a node mask to filter them out, thereby preventing their selection.

C. Agent

To effectively select target nodes that yield local approximations while satisfying error rate constraints, we propose a *functionality-aware agent*. This agent integrates the circuit representation learning model to capture both the structural and functional characteristics of the approximate circuit and the original circuit. This enables the agent to make functionally aware decisions during approximations.

We adopt *DeepGate2* [13] for learning representations of logic circuits and processing both the original circuit and the approximate circuit. This model produces both functional and structural gate-level embeddings that reflect each node's logic function and topological feature. To make global decisions, we generate two distinct graph-level embeddings:

- **Reference Embedding:** A graph-level representation of the original circuit, obtained by aggregating POs' gate embeddings using a multilayer perceptron (MLP). This embedding serves as an anchor for guiding approximations and enforcing functional correctness.
- **State Embedding:** A graph-level embedding of the approximate circuit, similarly obtained by aggregating the representations of the PO nodes. This embedding captures the cumulative structural and functional deviations from the original circuit.

At each timestep, the agent also computes **action embeddings** for all nodes. These are derived by concatenating each node's learned gate embeddings with a one-hot encoding of its gate type (i.e., primary input, AND gate, or NOT gate), followed by an MLP transformation and refinement using a self-attention mechanism. The self-attention mechanism enables the model to account for inter-node dependencies and gate context, improving prioritization of target candidates.

To ensure that the agent only selects meaningful actions, we apply a dynamic node masking mechanism to filter out invalid actions:

- All inactive nodes in the current approximate circuit are excluded.
- The node selected and evaluated in the previous timestep is also excluded to avoid a redundant approximation.

The RL agent uses reference embedding, state embedding, and action embeddings to produce a probability distribution over all active nodes. The agent then samples a target node in the distribution and passes it to the environment to apply LAC and estimate the error rate. For the critic network in PPO, we also use a circuit representation learning model with an MLP layer to estimate the advantage.

D. LAC with Infeasible Target Node Pruning

For each target node selected by the agent, we re-express the target node using new fanins to obtain the approximate circuit at the current state. Our strategy only selects candidate fanins from the TFI cone of the target node, preventing changing the functional support of the target node. We do not consider nodes with levels higher than the target node since this may create a dependency loop, which is not generally allowed in combinational circuits. For each candidate fanin set, we derive the approximate function for local approximation and apply

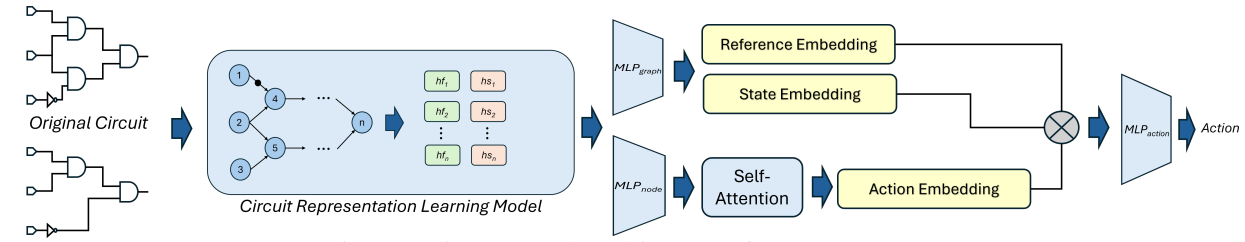

Fig. 2: Policy Network Architecture of Our RL Agent.

it to the current approximate circuit. The details of the LAC derivation will be elaborated in Section III-E.

To reduce the area of the approximate circuit, we have to turn as many active nodes into inactive ones as possible. Hence, the approximation selecting the two lowest-level nodes as fanins is the LAC that reduces the most area, regardless of the error introduced into the circuit. When the set of lowest-level fanins is unable to reduce circuit area, we consider the selected target node as infeasible. To accelerate this approximation process, we need to check the feasibility of the target node. If a target node is considered infeasible, we reselect another node until a feasible one is selected.

E. Environment

Our framework integrates an error estimation mechanism into a priority-based LAC selection strategy. This integration allows the environment to efficiently evaluate candidate LACs using parallel simulation and apply the most promising one under the error rate constraint for state transition.

1) Error Rate Estimation: For each LAC candidate associated with a given target node, we estimate the accuracy change in the circuit using a simulation-based analysis. Specifically, we reuse the simulation values from the previous stage, which are originally used to evaluate the actual error rate of the approximate circuit in the environment, to perform a local error rate estimation. Each LAC modifies the functionality of a target node based on a set of fanins and the associated approximate function. To estimate the error rate introduced by a candidate LAC, we locally simulate the approximate function using the simulation values of its fanins, yielding a new set of simulation values for the target node. These values are then compared with the original values of the target node in the previous state using the Hamming distance metric.

However, this comparison is restricted to the subset of input patterns, which are those that do not produce any errors in the POs under the current approximate circuit. We focus only on patterns that currently produce correct outputs, because changing their simulation values could introduce new errors in later iterations. In contrast, patterns that already cause PO errors will not produce new errors due to the current approximation, and in some cases may even cancel out existing errors, improving overall correctness. Moreover, if the local errors in the target node are not propagated to POs, this approximation will not introduce new errors in the POs either. Hence, when we compute the error rate at the POs, the actual error rate increment will not exceed the estimated error rate increment. In other words, the error is overestimated, ensuring that the actual error rate does not grow beyond our predictions.

Let p denote an input pattern and let $f(p)$ and $f_{\text{approx}}(p)$ denote the original function value and approximate function value on the target node with respect to p, respectively. We compare $f()$ and $f_{\text{approx}}()$ only on the subset S of all input patterns that do *not* cause any error in the POs of approximate circuit. Hence, the estimated error rate increment is calculated as follows:

$$\Delta_{\text{est}} = \frac{1}{|S|} \sum_{p \in S} \left[f(p) \oplus f_{\text{approx}}(p) \right]$$

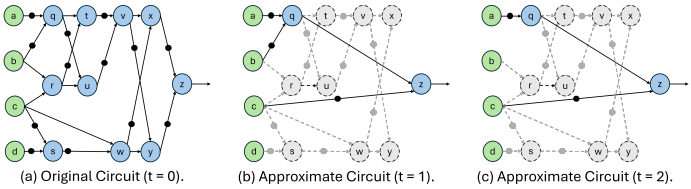

Fig. 3: An example of the LAC.

TABLE II: The truth table for the original circuit and approximate circuit in Fig. 3.

$abcd$	q	w	x	y	z	z_{approx}	$abcd$	q	w	x	y	z	z_{approx}
0000	1	0	0	0	1	1	1000	0	0	1	0	0	0
0001	1	0	0	0	1	1	1001	0	0	1	0	0	0
0010	1	1	0	1	0	0	**1010**	0	1	0	0	**1**	**0**
0011	1	1	0	1	0	0	**1011**	0	1	0	0	**1**	**0**
0100	0	0	1	0	0	0	1100	0	0	1	0	0	0
0101	0	0	1	0	0	0	1101	0	0	1	0	0	0
0110	0	1	0	1	0	0	1110	0	1	0	1	0	0
0111	0	1	0	1	0	0	1111	0	1	0	1	0	0

We consider the circuit in Fig. 3 as an illustrative example of our approach. Assume that the nodes a, b, c, and d in Fig. 3(a) are PIs, and the node z is the PO and is the first target node selected by the agent. In Fig. 3(b), the fanin nodes q and c are selected as the new fanins for z at time step $t = 1$, and the approximate function is derived as $z_{approx} = q \cdot c'$, which yields the minimum error rate increment.

For the approximate function, the error rate increment is determined as follows: given $z = x' \cdot y'$ and $z_{\text{approx}} = q \cdot c'$, we count the number of patterns that results in different values in z and z_{approx}. The truth table in Table II shows that the mismatches between z and z_{approx} occur in $abcd \in \{1010, 1011\}$, where $z = 1$ but $z_{\text{approx}} = 0$. Since the original circuit in Fig. 3(a) has no PO errors on any of the 16 patterns (i.e., S contains all patterns), the estimated error rate increment is $\Delta_{\text{est}} = 2/16 = 12.5\%$. If the error rate of the resulting approximate circuit is below the margin we set for the LAC prioritization technique, the approximate circuit is retained as the new state (S_{t+1}), and the agent proceeds to select new target nodes (A_{t+1}) for further approximation at the next time step $t+1$. Otherwise, the environment selects alternative fanin

nodes with higher levels or new target nodes to derive another LAC.

In this example, if we further approximate the circuit in Fig. 3(b), we can obtain the approximate circuit shown in Fig. 3(c). Assume that the node q is selected as the target node and the node a is selected as the new fanin to replace node b at the next time step $t = 2$. Then, the new approximate function for node q is derived as $q_{\text{approx}} = a' \cdot a' = a'$. For the PO, the new approximate function is derived as $z_{\text{approx}'} = a' \cdot c'$. Since the patterns $abcd \in \{1010, 1011\}$ already produce PO errors, they are excluded from S. In this case, the estimated error rate increment becomes $\Delta_{\text{est}} = 4/14 = 28.6\%$, i.e., counting the four patterns $abcd \in \{0100, 0101, 0110, 0111\}$ but divided by the size of the error-free subset of PI input patterns. This LAC corresponds to a real error rate increment of 12.5% since only the errors in $abcd \in \{0100, 0101\}$ are propagated to the PO, which does not exceeds the estimated error rate. Consequently, the error rate of the result approximate circuit in Fig. 3(c) equals to $12.5\% + 12.5\% = 25\%$, where the errors occur in $abcd \in \{0100, 0101, 1010, 1011\}$.

2) LAC Prioritization: In addition to error rate estimation, we propose an LAC prioritization scheme to derive the most promising LAC for a target node. For each candidate LAC of a target node, we first prioritize the selection of its fanin pairs with the lowest-level in the TFI cone of the target node. This is because such pairs are more likely to reduce area and delay without introducing cycles. For each selected pair of fanins, we then derive their approximate function for the succeeding error rate estimation.

The LAC derivation process is adaptive to the current error rate of the circuit. If the estimated error rate of the current circuit exceeds a predefined margin, which means that it might be close to the error rate constraint, we derive the LAC with the smallest error rate increment. Conversely, if the error rate is below the margin, we apply an LAC whose estimated error rate is less than the error rate constraint and the fanins' levels are the lowest, enabling the approximation on more area and delay optimization.

F. Reward Function

After applying an LAC to the target node, the environment assigns a reward comprising three parts: area reduction, exploration incentive, and penalty. If the circuit remains within the error-rate constraint, the reward equals the area reduction ratio. To promote exploration, an extra positive reward is given when the action is rarely selected. If the error constraint is violated, a negative reward penalizes the agent. While optimizing both area and delay under error constraints is desirable, incorporating all objectives in the reward increases complexity and hinders convergence. We therefore decouple these objectives: area reduction ratio is captured in the reward, whereas delay and error rate control are managed externally through LAC prioritization and error rate estimation (Section III-E). This separation simplifies the agent's learning task while still enabling an effective multi-objective optimization across the ALS.

G. Backtracking Mechanism

While our RL agent guides the approximation process through the selection of target nodes and LACs, it remains challenging to guarantee that all explored trajectories will lead to promising states of approximate circuits under a strict error rate constraint. To ensure that our ALS can recover from infeasible trajectories and continue making progress, we introduce a structured backtracking mechanism. In this backtracking mechanism, we define two parameters: M, the number of target nodes to be evaluated per state, and N, the number of LACs tried for each target node. For a given circuit state, the RL agent selects a candidate target node and attempts up to N LACs derived from that node. If all these LACs result in circuits violating the error rate constraint, the target node is considered infeasible, and the agent proceeds to the next target node candidate. This process continues until either a valid approximation is found or M target nodes have been evaluated. If no valid LAC is found for these M nodes, our approach backtracks to the previous feasible state. This rollback allows our ALS to escape from unproductive or overly constrained regions of the search space, and returns to a more promising configuration. This backtracking mechanism therefore plays a critical role in balancing objective optimization and constraint satisfaction, ensuring the ALS remains efficient even in large design spaces.

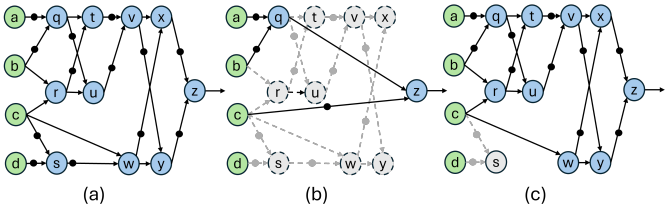

Fig. 4: An example of the proposed backtracking mechanism.

Consider the backtracking procedure in Fig. 4, where the target node z is replaced from its original function $z = x' \cdot y'$ in Fig. 4(a) to the new approximate function $z = q \cdot c'$ in Fig. 4(b). In this example, we assume that the error rate constraint is 10%. According to the Table II, the error rate increment equals to $\Delta_{\text{est}} = 2/16 = 12.5\%$. Although this LAC was selected because it has the minimum estimated error increment among all candidates at the time step t, the resulting error rate still exceeds the error rate constraint. According to the backtracking mechanism, we revert to the previous state S_t in Fig. 4(a), restoring z to its original function, and then continue the selection for a different target node.

Suppose the agent subsequently selects node w as the new target node. Using the same procedure, the environment derives a new approximate function for w. Since w was originally realized as $w = c \cdot s'$ with $s = c'd'$, we have $s' = c + d$, and thus $w = c(c + d) = c$. Therefore, the environment proposes the function $w = c$ for the new LAC using the error rate estimation approach, and the estimated error increment is 0. Consequently, this LAC is accepted and committed as the new S_{t+1} as illustrated in Fig. 4(c) since the real error rate increment does not exceeds the constraint.

IV. Experimental Results

We implement the proposed approach in Python using the PyTorch framework. Experiments were performed on a workstation with an Intel Core i7-14700K CPU, an NVIDIA

TABLE III: Experimental benchmarks.

Circuit	ADP	Circuit	ADP	Circuit	ADP
misex	546	c2670	13880	cps	24880
c880	8075	simple_spi	8150	c5315	49525
chkn	7224	c3540	39522	c7552	92220
c1908	13184	dalu	24541	alu4	38424

GeForce RTX4090 GPU, and a 128GB RAM, running on Ubuntu 24.04.2 LTS. Benchmark circuits were selected from IWLS2005 [1] and MCNC [17]. Each benchmark was initially transformed into the AIG format by ABC [15].

For the RL training phase, each RL agent is trained with a 5% error rate constraint, and the number of training episodes is set to 250. The learned policy is retained and reused across different error rate scenarios without further training, eliminating repeated training overhead when adjusting error rate constraint. We employ ADAM optimizers for both networks with 1×10^{-4} learning rates, 1×10^{-5} epsilon term, and 1×10^{-5} weight decay on the actor. To stabilize learning, we apply a cosine-annealing warm-restart scheduler that periodically resets the learning rate over 10 epochs with doubling cycles. DRL agent training uses the PPO and the policy updates are regulated by discount factor $\gamma = 0.98$ and GAE parameter $\lambda = 0.98$. We execute 10 epochs per batch with a trust-region-like clipping factor $\epsilon = 1$.

To accelerate the ALS flow, we adopt the *Node to Constant* method used in [6] [16] for pre-processing in the stage of backtracking, where nodes with high logic probabilities of 1 or 0 are replaced with constants 1 or 0 within a specified margin. The user-specified parameters of our approach are set as follows. The numbers M and N in the backtracking stage are both set to 3. Additionally, we set a 15-minute time limit when running the backtracking stage.

For each circuit, we measured the area (node count) and delay (level) and computed the corresponding ADP. Baselines include the genetic-algorithm method [6] and the node-merging method [16], which are the recent ALS methods that also evaluate the results in AIG representation. The reduction ratio of ADP was calculated with respect to the original benchmarks. The benchmark information is shown in Table III. Similarly to previous ALS [6] [16], we use 100,000 random input patterns under a uniform distribution to evaluate the error rates of approximate circuits. The error rate constraint in POs is set to 5% and 10%, which is the same as [6] and [16]. Additionally, for the LAC prioritization technique, we set the error rate margin to 0.5%. That is, if the difference between the error rate in the approximate circuit and the error rate constraint is within 0.5%, our approach always selects the LAC with the minimum estimated error rate increment.

In our evaluation, the same RL policy trained under the 5% error rate constraint was reused across different error rate scenarios, demonstrating the generalization and reusability of the learned policy. The 5% results represent the best circuits found during training, while the 10% results are derived by applying the learned policy during the backtracking phase under a relaxed error rate constraint. This highlights the efficiency and flexibility of decoupling policy learning from adjustment to different error rate constraints. The comparison of results is summarized in Table IV and V. Under the 5% error rate constraint (Table IV), our approach consistently outperforms the results in both the [16] and [6]. On average, we achieve a 56.79% reduction in ADP, compared to 38.56% in [16] and 52.51% in [6]. This corresponds to a 47% improvement over [16] and a 8% gain over [6]. Notably, for benchmarks *chkn*, *c1908*, and *cps*, our approach achieves reduction ratio exceeding 80%. When the error rate constraint is relaxed to 10%, our approach further increases its performances, achieving a 63.53% average ADP reduction, outperforming [16] and [6] by factors of 37% and 7%, respectively. A key advantage of our approach is its joint optimization of area and delay. While previous methods focus largely on the area, our framework applies an LAC with a reward function on area metric and an LAC prioritization for delay optimization.

TABLE IV: The comparison of experimental results among [16], [6], and ours under a 5% error rate constraint.

Circuit	[16]		[6]		Ours	
	ADP	ADP Red. Ratio	ADP	ADP Red. Ratio	ADP	ADP Red. Ratio
misex	228	58.24%	192	64.84%	164	**69.96%**
c880	6675	17.34%	6325	21.67%	5149	**36.24%**
chkn	858	88.12%	816	88.70%	518	**92.83%**
c1908	3703	71.91%	2432	81.55%	1650	**87.48%**
c2670	10800	22.19%	7140	48.56%	6015	**56.66%**
simple_spi	6450	20.86%	5625	**30.98%**	7150	12.27%
c3540	36608	7.37%	29750	24.73%	24459	**38.11%**
dalu	12708	**48.22%**	18656	23.98%	15336	37.51%
cps	6240	74.92%	3560	**85.69%**	4470	82.03%
c5315	47808	3.47%	32508	34.36%	31475	**36.45%**
c7552	86925	5.74%	42315	54.12%	28534	**69.06%**
alu4	21408	44.29%	11167	**70.94%**	14252	62.91%
Average	–	38.56%	–	52.51%	–	**56.79%**
Ratio	–	1	–	1.36	–	**1.47**

TABLE V: The comparison of experimental results among [16], [6], and ours under a 10% error rate constraint.

Circuit	[16]		[6]		Ours	
	ADP	ADP Red. Ratio	ADP	ADP Red. Ratio	ADP	ADP Red. Ratio
misex	156	71.43%	96	**82.42%**	100	81.68%
c880	6375	21.05%	5832	27.78%	4636	**42.59%**
chkn	324	**95.51%**	672	90.70%	328	95.46%
c1908	1683	87.23%	1650	**87.48%**	1650	**87.48%**
c2670	8960	35.45%	6953	49.91%	5222	**62.38%**
simple_spi	6100	25.15%	5409	**33.63%**	5922	27.34%
c3540	32718	17.22%	27192	31.20%	21240	**46.26%**
dalu	10314	**57.97%**	13960	43.12%	12402	49.46%
cps	6016	75.82%	3320	**86.66%**	3580	85.61%
c5315	46620	5.87%	30108	**39.21%**	31225	36.95%
c7552	83082	9.91%	36040	60.92%	28138	**69.49%**
alu4	17549	54.33%	9152	76.18%	8580	**77.67%**
Average	–	46.41%	–	59.10%	–	**63.53%**
Ratio	–	1	–	1.27	–	**1.37**

V. Conclusion

In this paper, we propose a novel DRL-based framework for approximate logic synthesis under error rate constraints. By integrating a functionality-aware agent, structured LAC selection, localized error estimation, and a backtracking mechanism, our approach effectively balances area, delay, and accuracy. Experimental results demonstrate that our approach achieves substantial reductions in ADP, outperforming the previous methods.

REFERENCES

[1] C. Albrecht. (2005) Iwls 2005 benchmarks. [Online]. Available: https://iwls.org/iwls2005/benchmarks.html

[2] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *2013 18th IEEE European Test Symposium (ETS)*, 2013, pp. 1–6.

[3] Y. Kim, Y. Zhang, and P. Li, "An energy efficient approximate adder with carry skip for error resilient neuromorphic vlsi systems," in *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2013, pp. 130–137.

[4] P. Kulkarni, P. Gupta, and M. Ercegovac, "Trading accuracy for power with an underdesigned multiplier architecture," in *2011 24th Internatioal Conference on VLSI Design*, 2011, pp. 346–351.

[5] Y.-A. Lai, C.-C. Lin, C.-C. Wu, Y.-C. Chen, and C.-Y. Wang, "Efficient synthesis of approximate threshold logic circuits with an error rate guarantee," in *2018 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, 2018, pp. 773–778.

[6] C.-T. Lee, Y.-T. Li, Y.-C. Chen, and C.-Y. Wang, "Approximate logic synthesis by genetic algorithm with an error rate guarantee," in *2023 28th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2023, pp. 146–151.

[7] Y.-T. Li, I. Chen, Y.-C. Chen, and C.-Y. Wang, "Approximate logic synthesis for dot-inverter graphs using node merging-enhanced genetic algorithm-based approach," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2025.

[8] J. Liang, J. Han, and F. Lombardi, "New metrics for the reliability of approximate and probabilistic adders," *IEEE Transactions on Computers*, vol. 62, no. 9, pp. 1760–1771, 2013.

[9] C. Meng, A. Mishchenko, W. Qian, and G. D. Micheli, "Efficient resubstitution-based approximate logic synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pp. 1–1, 2024.

[10] C. Meng, W. Qian, and A. Mishchenko, "Alsrac: Approximate logic synthesis by resubstitution with approximate care set," in *2020 57th ACM/IEEE Design Automation Conference (DAC)*, 2020, pp. 1–6.

[11] G. Pasandi, S. Nazarian, and M. Pedram, "Approximate logic synthesis: A reinforcement learning-based technology mapping approach," in *20th International Symposium on Quality Electronic Design (ISQED)*, 2019, pp. 26–32.

[12] I. Scarabottolo, G. Ansaloni, G. A. Constantinides, and L. Pozzi, "Partition and propagate: an error derivation algorithm for the design of approximate circuits," in *2019 56th ACM/IEEE Design Automation Conference (DAC)*, 2019, pp. 1–6.

[13] 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)*, 2023, pp. 1–9.

[14] S. Su, C. Zou, W. Kong, J. Han, and W. Qian, "A novel heuristic search method for two-level approximate logic synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 3, pp. 654–669, 2020.

[15] B. L. Synthesis and V. Group. (2025) Abc: A system for sequential synthesis and verification. [Online]. Available: https://people.eecs.berkeley.edu/~alanmi/abc

[16] K. S. Tam, C.-C. Lin, Y.-C. Chen, and C.-Y. Wang, "An efficient approximate node merging with an error rate guarantee," in *2021 26th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2021, pp. 266–271.

[17] S. Yang, "Logic synthesis and optimization benchmarks user guide version 3.0," Microelectronics Center of North Carolina, Technical Report, 1991, accessed: 2025-05-25. [Online]. Available: https://ddd.fit.cvut.cz/www/prj/Benchmarks/LGSynth91.pdf