A Hierarchical Adaptive Multi-Task Reinforcement Learning Framework for Multiplier Circuit Design

Zhihai Wang [1] **Jie Wang** [1 ✉] **Dongsheng Zuo** [2] **Yunjie Ji** [1] **Xilin Xia** [1] **Yuzhe Ma** [2] **Jianye Hao** [3 4]
Mingxuan Yuan [3] **Yongdong Zhang** [1] **Feng Wu** [1]

Abstract

Multiplier design—which aims to explore a large combinatorial design space to simultaneously optimize multiple conflicting objectives—is a fundamental problem in the integrated circuits industry. Although traditional approaches tackle the multi-objective multiplier optimization problem by manually designed heuristics, reinforcement learning (RL) offers a promising approach to discover high-speed and area-efficient multipliers. However, the existing RL-based methods struggle to find Pareto-optimal circuit designs for all possible preferences, i.e., weights over objectives, in a sample-efficient manner. To address this challenge, we propose a novel **h**ier**a**rchical adapti**ve** (HAVE) multi-task reinforcement learning framework. The *hierarchical* framework consists of a meta-agent to generate *diverse* multiplier preferences, and an adaptive multi-task agent to *collaboratively* optimize multipliers conditioned on the dynamic preferences given by the meta-agent. To the best of our knowledge, HAVE is *the first* to well approximate Pareto-optimal circuit designs for the entire preference space with high sample efficiency. Experiments on multipliers across a wide range of input widths demonstrate that HAVE significantly Pareto-dominates state-of-the-art approaches, achieving up to 28% larger hypervolume. Moreover, experiments demonstrate that multipliers designed by HAVE can well generalize to large-scale computation-intensive circuits.

This work was done when Zhihai Wang was an intern at Huawei. [1]CAS Key Laboratory of Technology in GIPAS & MoE Key Laboratory of Brain-inspired Intelligent Perception and Cognition, University of Science and Technology of China [2]Microelectronics Thrust, Hong Kong University of Science and Technology (Guangzhou) [3]Noah's Ark Lab, Huawei Technologies [4]College of Intelligence and Computing, Tianjin University. Correspondence to: Jie Wang <jiewangx@ustc.edu.cn>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024.

1. Introduction

Arithmetic logic circuits are the most elementary building blocks in many real-world circuits, such as the central processing units and graphics processing units (Holdsworth, 1987; Das et al., 2019). The multiplication operation is one of the most fundamental and frequently used arithmetic operations in many computation-intensive applications, such as deep neural networks (DNNs), digital signal processors, and microprocessors (Hashemian, 2002; Elguibaly, 2000; Zuo et al., 2023). In particular, the multiplication operation accounts for more than 90% of all operations in many popular DNNs, including ResNet (He et al., 2016), ViT (Dosovitskiy et al., 2021), and BERT (Devlin et al., 2019). Therefore, designing high-speed and area-efficient multipliers is critical for boosting the performance of computation-intensive applications, especially for DNN accelerators.

However, multiplier design is a challenging combinatorial optimization problem with multiple conflicting objectives, which can be extremely hard to solve due to its $\mathcal{NP}$-hard nature (Hillar & Lim, 2013; Song et al., 2022). To optimize multipliers, many traditional approaches rely on manually designed heuristics to minimize the analytical evaluation metrics such as circuit size (Roy et al., 2021). However, they can exhibit poor performance subsequent to synthesis due to the significant gap between their analytical evaluation metrics and physical synthesis metrics (Roy et al., 2021; Zuo et al., 2023). To bridge this gap, recent approaches (Roy et al., 2021; Song et al., 2022; Zuo et al., 2023) use reinforcement learning (RL) to optimize circuits with synthesis in the loop, which offers a promising avenue for designing high-speed and area-efficient circuits.

However, the existing RL-based methods (Roy et al., 2021; Zuo et al., 2023) struggle to find Pareto-optimal circuit designs for all possible preferences (i.e., weights over objectives) in a sample-efficient manner due to the following reasons. First, they circumvent the multi-objective property, and use scalarized objectives weighted by predetermined preferences to train many individual agents for optimizing multipliers. This strictly limits the knowledge sharing among different but related optimization problems. Second, they struggle to sufficiently explore the large design space,

as it grows exponentially with the input bit widths of multipliers (Roy et al., 2021). Third, obtaining the reward signal with synthesis in the loop is highly time-consuming, thereby leading to significant demands on sample efficiency.

To address this challenge, we propose a novel **h**ier**a**rchical adapti**ve** (HAVE) multi-task reinforcement learning (MTRL) framework for discovering high-speed and area-efficient multiplier designs. To the best of our knowledge, HAVE is *the first* to well approximate Pareto-optimal circuit designs that covers the entire multiplier preference space in a sample-efficient manner by proposing a hierarchical framework. The framework is comprised of (1) a meta-agent to generate multiple *diverse* preferences, (2) and an adaptive multi-task agent to *collaboratively* optimize multipliers conditioned on the dynamic preferences. The adaptive multi-task agent exploits common properties across related optimization problems, significantly improving sample efficiency. Moreover, an appealing feature of HAVE is a factored Q-function representation, which significantly boosts sample efficiency by leveraging the underlying combinatorial structure of action spaces to decompose the Q-function into multiple independent components.

We evaluate HAVE by designing multipliers across a wide range of input widths. Experiments demonstrate that HAVE discovers state-of-the-art designs that Pareto-dominate the multipliers designed by human-based, mathematical optimization-based, and learning-based baselines, achieving up to 28% larger hypervolume. Moreover, we deploy multipliers designed by HAVE and baselines into large-scale computation-intensive circuits, and experiments show that HAVE significantly outperforms the baselines in terms of both area and delay. Our results demonstrate the superior ability to discover high-speed and area-efficient circuits with HAVE in real-world computing circuits.

We summarize our major contributions for both the RL community and hardware design community as follows. (1) To the best of our knowledge, HAVE is **the first** to well approximate Pareto-optimal circuit designs that covers the entire multiplier preference space in a highly sample-efficient manner, which is significant for the application of RL methods to designing diverse multipliers. (2) We propose a novel **hierarchical** framework to decompose the task of exploring multiplier preference space and searching the design space into two hierarchies, which effectively **leverages the structure** between the preference space and the design space for significantly boosting sample efficiency. (3) We propose an adaptive multi-task reinforcement learning (RL) agent to collaboratively search the design space of multipliers conditioned on the dynamic preferences given by the meta agent. The novel adaptive multi-task formulation allows us to **exploit common properties** across related search problems, which is significant for improving sample efficiency. (4) We propose to leverage the underlying combinatorial structure of action spaces via a factored Q-function representation, which further improves sample efficiency. Moreover, we are **the first** to theoretically show that the factored Q-learning algorithm can approximate the optimal Q-function within an error bound. (5) Experiments on multipliers across a wide range of input widths demonstrate that HAVE significantly Pareto-dominates state-of-the-art approaches, achieving up to **28%** larger hypervolume.

2. Related Work

In this section, we discuss related work on multi-objective reinforcement learning (MORL), and defer additional related work to Appendix B due to limited space.

To approximate a set of Pareto-optimal solutions, the most widely used approach is to repeatedly perform a single-policy algorithm over various preferences (Yang et al., 2019; Basaklar et al., 2023). Although the existing RL for circuit optimization methods (Roy et al., 2021; Song et al., 2022; Zuo et al., 2023) leverage this approach, they suffer from poor sample efficiency and computational efficiency. Recent MORL work (Yang et al., 2019; Basaklar et al., 2023) proposes to use a multi-objective conditioned Q-learning approach to simultaneously learn a set of policies over multiple preferences. Although our method uses a multi-objective conditioned Q-network as well, HAVE maintains the following major contributions over previous MORL methods. First, HAVE learns a meta-agent to explore multiple diverse preferences at each step, which is more sample-efficient compared to existing random sampling strategy. Second, HAVE formulates the optimization problem under multiple dynamic preferences as an adaptive MTRL, which carries the potential to exploit common properties among tasks for significantly improving sample efficiency. Third, HAVE leverages a factored Q-value representation, which leverages the underlying structure of action spaces and thus significantly improves sample efficiency.

Multiplier Circuit Design In general, the approaches for multiplier circuit design fall into three categories as follows. First, manual designs (Wallace, 1964; Dadda, 1983; Itoh et al., 2005) are obtained by leveraging human expertise architectures from regular architectures which require high engineering effort. Second, traditional algorithmic methods (Xiao et al., 2021; Liu et al., 2003; Roy et al., 2013) generate circuit architectures based on particular strategies, such as mathematical programming and heuristic search. However, they rely on proxy metrics to optimize circuits, such as the size and/or depth of a circuit, which results in a gap with real end flow. Third, recent methods (Zuo et al., 2023; Roy et al., 2021; Song et al., 2022) propose to use reinforcement learning to optimize circuits based on the actual evaluation metrics within the optimization loop, which offers promising approaches to bridge the gap of proxy metrics. However,

the existing methods suffer from inefficient exploration and thus tend to find sub-optimal designs due to the vast search space and the intricate balance of multiple conflicting objectives. Moreover, recent work (Song et al., 2023) proposes to use generative models to generate adders. Recent efforts also study leveraging machine learning to directly design circuits (Google DeepMind, 2023; Li et al., 2023; 2024a).

3. Background

In this section, we introduce the multiplier architecture, the concept of Pareto optimal, and the state-of-the-art RL for multiplier design method, i.e., RL-MUL (Zuo et al., 2023).

Multiplier Architecture The multiplier consists of three parts: a partial product generator (PPG), a compressor tree (CT), and a carry propagation adder (CPA) (Xiao et al., 2021). The PPG produces partial products (PPs) from the multiplicand and multiplier. The CT aims to compress the PPs to two rows in parallel. After the compression process, the CPA sums up the two rows of PPs to get the final product. As the CT usually dominates the delay and area of a multiplier (Xiao et al., 2021; Zuo et al., 2023), RL-MUL focuses on optimizing the CT structure of a multiplier. Please refer to Appendix C for details of the architecture.

Pareto Optimal In an n-objective optimization problem, we say a solution x Pareto dominates another y if $\forall i \in [1, n], f_i(x) \leq f_i(y) \wedge \exists i \in [1, n], f_i(x) < f_i(y)$. Given a solution set, the Pareto front consists of Pareto-optimal solutions that are not dominated by any other solution. To evaluate the quality of the Pareto front, we apply a hypervolume metric (see Appendix C for definition). Note that a Pareto front with a larger hypervolume is deemed superior.

RL for Multiplier Design RL-MUL starts from a given initial compressor tree solution, and iteratively refines the compressor tree structure. In terms of the **state space**, RL-MUL represents each state, i.e., a compressor tree solution, as a three-dimensional image. In terms of the **action space**, RL-MUL designs four types of action to refine a compressor tree in a given column. Then, RL-MUL defines the action space as a discrete space composed of $4 \times N_C$ discrete actions, where N_C denotes the number of columns. Each action $i \in [0, 1, \ldots, (4 \times N_C - 1)]$ is represented by executing the j-th action type at the k-th column, where $j = i \pmod 4$ and $k = \lfloor \frac{i}{4} \rfloor$. In terms of the **reward function**, RL-MUL first invokes a synthesis tool to obtain the area and delay of the designed multiplier at each step. Then, RL-MUL designs a reward r_t as the difference between the area (delay) of the multiplier at step $t-1$ and that at step t. In terms of **training**, RL-MUL leverages the deep Q-network algorithm (Mnih et al., 2015) to train policies. We defer details of the method to Appendix C.

4. A Hierarchical Adaptive Multi-Task Reinforcement Learning Framework

In this section, we present our proposed **h**ier**a**rchical adapti**ve** (HAVE) multi-task reinforcement learning framework. We first present an overview of HAVE as follows.

To efficiently approximate Pareto-optimal circuit designs for the entire multiplier preference space, we propose a novel two-stage hierarchical framework. The framework consists of two agents working at different time-scales, where a meta-agent sufficiently explores the multiplier preference space to maximize the Pareto curves, and a lower-level agent optimizes multipliers conditioned on the dynamic preferences given by the meta-agent (see Figure 5 in Appendix E.1).

To sufficiently explore the multiplier preference space, we first propose a delay constraints augmented preference space. The augmented space enhances the traditional linear weight preference space by taking into account the synthesis-related delay constraints. Then, we formulate the preference generation problem as a markov decision process (MDP) to learn policies for generating multiple diverse preferences with the goal of maximizing Pareto curves (see Figure 1).

To optimize multipliers conditioned on the dynamic preferences given by the meta-agent, HAVE formulates the problem as an adaptive multi-task reinforcement learning (MTRL) problem. Each task corresponds to a circuit optimization problem conditioned on each preference generated by the meta-agent, respectively. Then, HAVE can leverage advanced MTRL algorithms to exploit common properties across these tasks for improving sample efficiency.

To solve the adaptive MTRL problem, HAVE proposes a factored multi-task conditioned deep Q-network. The conditioned network takes states and preferences as inputs, thus enabling generalizing across dynamic preferences. Moreover, the factored Q-function representation leverages the underlying structure of action spaces to decompose Q-functions, thus significantly improving sample efficiency.

4.1. Pareto-Driven Meta-Agent Learning

Delay Constraints Augmented Preference Space Many existing MORL methods (Yang et al., 2019; Abels et al., 2019b; Basaklar et al., 2023) maximize a joint objective function weighted by a preference vector to balance conflicting objectives, where the preference space comprises all possible weight vectors. However, we found that delay constraints for synthesis significantly impact preferences between objectives as well in the circuit optimization problem as shown in Figure 6 in Appendix G. In the context of circuit design, employing synthesis with tighter delay constraints signifies a stronger emphasis on optimizing for reduced delay. Although existing RL-based circuit optimization methods (Zuo et al., 2023; Roy et al., 2021) take

into account the delay constraints, they statically assign fixed delay constraints, which can lead to insufficient exploration and computational inefficiency. To address this challenge, we propose a delay constraints augmented preference space, which comprises the compositional vectors of weight vectors and delay constraints. However, incorporating the delay constraints into the preference space poses significant challenges for efficient exploration due to two reasons (see Appendix E.2). To sufficiently and efficiently explore the augmented preference space, we formulate the preference generation problem as an MDP, and learn Pareto-driven policies that generate multiple diverse preferences at each step to maximize Pareto-curves.

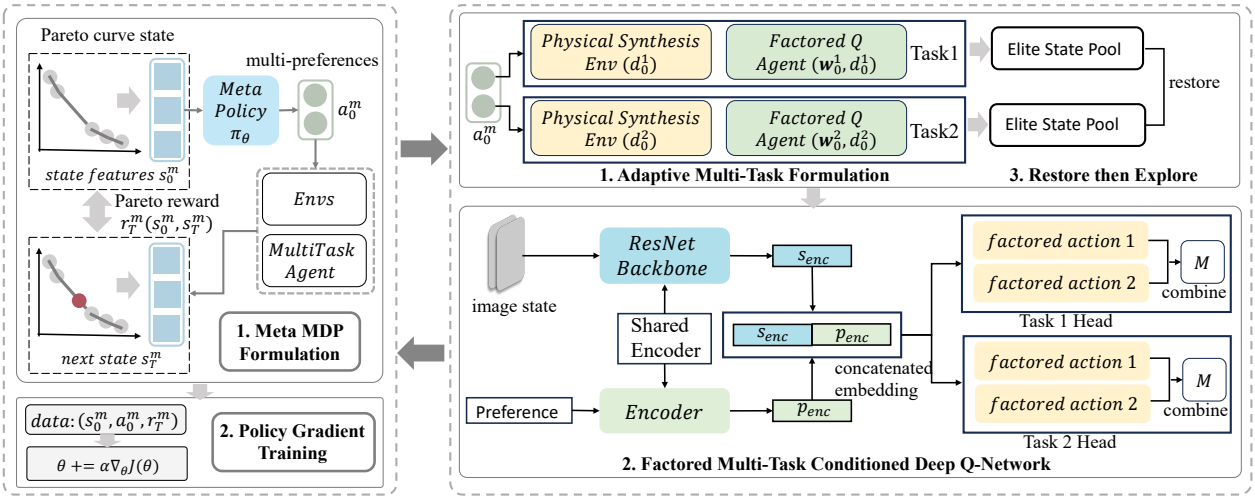
Meta Policy Learning We consider the meta MDP defined by the tuple $\mathcal{M}_m = (\mathcal{S}_m, \mathcal{A}_m, r_m, f_m)$. Specifically, we specify the meta state space $\mathcal{S}_m$, the meta action space $\mathcal{A}_m$, the meta reward function $r_m : \mathcal{S}_m \times \mathcal{A}_m \times \mathcal{S}_m \to \mathbb{R}$, and the meta transition function f_m in the following. **(1) The meta state space $\mathcal{S}_m$.** We design the meta state to track the evolving Pareto frontier to adaptively modify the generated user preferences for guiding the multi-task agent to collaboratively explore those under-explored regions. To encode the information of the evolving Pareto frontier, we manually design feature vectors. We present details in Appendix E.3. **(2) The meta action space $\mathcal{A}_m$.** To generate a set of N preferences, it is natural to define the action space by $(\mathcal{W} \times \mathcal{D})^N$, where $\mathcal{W} = [0, 1]^2$ denotes the space of linear weight vectors and $\mathcal{D}$ denotes the delay constraints for synthesis. However, the action space is large due to its combinatorial structure, and the meta-agent may generate numerous redundant preferences that exhibit similarity. To reduce redundant preferences generation, we decompose the entire preference space into N pairwise disjoint subspaces $\{\mathcal{A}_m^i = (\mathcal{W}^i \times \mathcal{D}^i)\}_{i=1}^N$, where $\mathcal{W} = \bigcup_{i=1}^N \mathcal{W}^i$ and $\mathcal{D} = \bigcup_{i=1}^N \mathcal{D}^i$. Then, we define the meta action space by $\mathcal{A}_m = \mathcal{A}_m^1 \times \cdots \times \mathcal{A}_m^N$. That is, the meta-agent samples N preferences from the N decomposed preference spaces, which can be diverse while complementary with each other. **(3) The meta Pareto-driven reward function r_m.** To encourage the meta-agent to generate pareto-driven preferences, we design the reward $r(s_t^m, a_t^m, s_{t+1}^m)$ by the difference between the hypervolume of the Pareto front at the next state s_{t+1}^m and that at the current state s_t^m. Thus, the meta-agent learns to generate diverse preferences that can maximize the Pareto curves. **(4) The transition function.** The transition function maps the current state and the action to the next state, which depends on the lower-level agent. To train the meta policy, we use a widely used reinforce algorithm (Sutton et al., 1999; Sutton & Barto, 2018). We provide implementation details in Appendix E.

4.2. Adaptive Multi-Task Agent Learning

(a) Adaptive Multi-Task RL Formulation To optimize multiplier circuits with multiple conflicting objectives conditioned on multiple preferences generated by the meta-agent, the existing methods (Zuo et al., 2023; Roy et al., 2021) use multiple scalarized objectives weighted by the preferences to train many individual agents to independently optimize circuits. However, they neglect the shared common properties among related optimization problems, leading to poor sample efficiency. To address this challenge, we propose to formulate the multiplier optimization problem conditioned on *dynamic* preferences as an *adaptive* multi-task RL problem. Specifically, we consider the adaptive multi-task MDP by the tuple $\mathcal{M}(\mathbf{w}_i, d_i) = (\mathcal{S}, \mathcal{A}, P, \gamma, \{\mathbf{r}_i, \mathbf{w}_i, d_i\}_{i=1}^N)$ that is conditioned on the dynamic preferences $(\mathbf{w}_i, d_i)_{i=1}^N$ given by the meta-agent. As mentioned in Section 3, we follow Zuo et al. (2023) to formulate the states as three-dimensional images, the action space as a discrete integer space, the transition function as a state refinement function with manually-designed legalization process. The major differences between the previous formulation and ours lie in the reward functions. First, the vectorized reward function remains invariant across preferences in the previous formulation. However, our vectorized reward function varies significantly across preferences due to the delay constraints. Second, the preferences in the previous formulation are fixed, while the preferences $\{(\mathbf{w}_i, d_i)\}_{i=1}^N$ in our formulation dynamically evolve. This significantly impacts the reward functions and thus affects the optimization objectives. The two differences pose significant challenges for learning in our formulation.

(b) Multi-Task Conditioned Deep Q-Network To approximate the Pareto-optimal circuit designs that adapts to the dynamic preferences using a single network, we propose a multi-task conditioned deep Q-network.

Shared Conditioned Encoder To learn Q-values that can generalize across diverse dynamic preferences, we model the encoder as a conditioned network, which is inspired by the universal value function approximators (Schaul et al., 2015; Abels et al., 2019a). Specifically, the encoder takes a state and a preference as an input, and outputs a joint concatenated embedding of the state and preference for Q-value prediction. To encode the three-dimensional image state, we employ the ResNet-18, which is popular in computer vision (He et al., 2016). To encode the preference vector, we can use any parametric neural network. For simplicity, we propose a non-parametric normalize layer to get the representation of preference vectors. Finally, we concatenate the state embedding and the preference embedding to achieve a joint embedding for Q-value prediction. Moreover, we propose to use a shared encoder network across multiple tasks, which can effectively improve sample efficiency by capturing common representations with significantly fewer trainable parameters compared to prior approaches (Roy



(a) Pareto-Driven **Meta-Agent** Learning (b) Adaptive **Multi-Task Agent** Learning

Figure 1. HAVE is a hierarchical framework comprising a Pareto-driven meta-agent and an adaptive multi-task agent. The meta-agent learns to generate diverse while complementary preferences with the goal of maximizing Pareto curves. The adaptive multi-task agent optimizes multipliers via a factored multi-task conditioned Q-learning algorithm and a restore-then-explore exploration strategy.

et al., 2021; Zuo et al., 2023; Song et al., 2022).

Multi-Task Decoder To learn Q-value functions across diverse tasks simultaneously, we propose a multi-task decoder, which parameterizes the decoder via a multi-head neural network. Each task head Q_{θ_i} approximates the Q-value function under the task environment conditioned on the i-th preference. Moreover, each task head predicts multi-objective Q-values, which is widely used in previous work (Roy et al., 2021; Yang et al., 2019; Basaklar et al., 2023).

Asynchronous Multi-Task Q-Learning To exploit common knowledge across tasks, we further propose a shared replay memory that stores all transitions collected from all task environments. However, the transition $(s, a, \mathbf{r}_j, \mathbf{w}_j, d_j)$ collected from the j-th task environment cannot be directly used to train the i-th task agent due to the preference difference. Thus, we propose a data relabeling mechanism, which transforms the transition $(s, a, \mathbf{r}_j, \mathbf{w}_j, d_j)$ into $(s, a, \mathbf{r}_j, \mathbf{w}_i, d_j)$ for training the i-th task agent. As a result, the multi-task agents can well collaboratively share knowledge on the design space across these task agents.

Moreover, to efficiently train the multi-task Q-network, we propose an asynchronous multi-task Q-learning algorithm. The algorithm asynchronously execute the multi-task agent in parallel on multiple instances of the environment conditioned on generated preferences. Specifically, each thread interacts with the multiplier design environment conditioned on each preference, and computes the gradient of the Q-learning loss at each step. We use a shared and gradually evolving target network in computing the Q-learning loss, following the approach proposed in the asynchronous Q-learning method (Mnih et al., 2016). Thus, for each step within the i-th environment, we update the Q-network to minimize the Bellman residual with a batch of transitions $\{(s_t, a_t, r, s_{t+1})\}_{t=1}^{M}$ sampled from the shared replay memory, i.e., $l(\theta_i) = \sum_{j=1}^{M} \frac{1}{M} \left[Q_{\theta_i}(s_j, a_j) - y(r_j, s_{j+1})\right]^2$, where $y(r_j, s_{j+1}) = r_j + \gamma \arg\max_a Q_{\bar{\theta}_i}(s_{j+1}, a)$.

(c) Factored Q-Value Update (FQA) Existing methods (Roy et al., 2021; Song et al., 2022; Zuo et al., 2023) directly represent the action space as a *flattened* set of non-negative integers ranging from zero to $4 * N_C - 1$ as mentioned in Section 3. However, they neglect the inherent combinatorial structure of the action space, wherein each action comprises a combination of column and type actions. To leverage the underlying compositional structure for improving sample efficiency, we leverage a factored action representation, which decomposes the original action space into two independent sub-action spaces, i.e., the column and type action spaces. Then, we represent the Q-value function as a combination of the factored Q-value functions defined on the factored action space. That is, we define the Q-value function by $Q(s, \mathbf{a}) = M(Q_1(s, a^1), Q_2(s, a^2))$, where M is a combination function and $\mathbf{a} = (a^1, a^2)$. To parameterize our factored Q-value functions, we first propose to instantiate the combination function via a simple and differentiable non-parametric summation function. Then, we parameterize the factored Q-value functions by $Q(s, \mathbf{a}) = \sum_{i=1}^{2} Q_{\theta_i}(s, a^i)$. Finally, we propose a factored Q-value update (FQA) scheme following the Bellman residual minimization, i.e., $l([\theta_1, \theta_2]) = \sum_{i=1}^{N} \frac{1}{N} \left[\sum_{j=1}^{2} Q_{\theta_j}(s_i, a_i^j) - y(r_i, s_{i+1})\right]^2$, where $y(r_i, s_{i+1}) = r_i + \gamma \arg\max_{\mathbf{a}} \sum_{j=1}^{2} Q_{\bar{\theta}_j}(s_{i+1}, a^j)$.

Advantages We discuss two potential advantages of FQA as follows. First, the Q-value function decomposition mechanism is analogous to decomposing a vector in terms of

some orthogonal basis vectors, which is beneficial for removing redundant information. Second, the FQA scheme allows the agent to update Q-values of multiple compositional actions using one sample (s_i, a_i, r_i, s_{i+1}), which can significantly improve sample efficiency (Sharma et al., 2017). Despite these potential advantages, existing convergence results from Bellman optimality operator no longer hold due to the factored Q-value function representation. Thus, we provide a theoretical analysis of FQA as follows.

Convergence Analysis of Tabular FQA We assume the state space, action space, and rewards are finite, which is commonly used in theoretical analysis of RL (Sutton & Barto, 2018). Thus, we can express the reward function and Q-value function in a vectorized form, i.e., $\mathbf{R}, \mathbf{Q} \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|\times 1}$ with items $\mathbf{R}_{(s,\mathbf{a})} = \mathbf{w}_i^T \mathbf{r}(s, \mathbf{a}|d_i)$ and $\mathbf{Q}_{(s,\mathbf{a})} = Q(s, \mathbf{a}; \mathbf{w}_i, d_i)$. We assume that the reward function is bounded, i.e., $|\mathbf{w}^T \mathbf{r}(s, \mathbf{a}|d)| \leq R, \forall s \in \mathcal{S}, \mathbf{a} \in \mathcal{A}$. Note that the MORL problem degenerates to a single-objective RL problem when given a generated preference $(\mathbf{w}_i, d_i)$. For simplicity, we omit the preference $(\mathbf{w}_i, d_i)$ in the following. The transition matrix $\mathbf{P} \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}|\times|\mathcal{S}|}$ is defined by $\mathbf{P}_{(s,\mathbf{a}),(s')} = P(s'|s, \mathbf{a})$. Notice that subscript $(s, \mathbf{a})$ is an abbreviation for index $|\mathcal{A}| \times [s] + [\mathbf{a}]$ where notation $[s]$ denotes the index of s in the discrete space $\mathcal{S}$ and so does $[\mathbf{a}]$. The standard Q-learning (Watkins & Dayan, 1992) algorithm updates Q-value based on the Bellman optimality operator $\mathcal{T}\colon \mathbb{R}^{|\mathcal{S}||\mathcal{A}|\times 1} \to \mathbb{R}^{|\mathcal{S}||\mathcal{A}|\times 1}$, i.e., $\mathcal{T}[\mathbf{Q}] = \mathbf{R} + \gamma \mathbf{P}\mathcal{H}[\mathbf{Q}]$. The operator $\mathcal{H}\colon \mathbb{R}^{|\mathcal{S}||\mathcal{A}|\times 1} \to \mathbb{R}^{|\mathcal{S}|\times 1}$ is defined by $\mathcal{H}[\mathbf{Q}]_{(s)} = \max_{\mathbf{a}\in\mathcal{A}} \mathbf{Q}_{(s,\mathbf{a})}$.

As the aforementioned factored Q-value function representation, we define the Q-value function $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ by a linear summation of the sub-Q functions $\tilde{Q}_l : \mathcal{S} \times \mathcal{A}_n \to \mathbb{R}, l = 1, \ldots, L$. That is, $Q(s, \mathbf{a}) := \sum_{j=1}^{L} \tilde{Q}_j(s, a^j)$. We also express the sub-Q functions $\tilde{Q}_n$ in a vectorized form i.e., $\tilde{\mathbf{Q}}_l \in \mathbb{R}^{|\mathcal{S}||\mathcal{A}_l|\times 1}$ with items $\tilde{\mathbf{Q}}_{l(s,a^l)} := \tilde{Q}_l(s, a^l)$. Note that the domains of different sub-Q functions are different. To facilitate convergence analysis, we expand the domain of $\tilde{Q}_l$ from $\mathcal{S} \times \mathcal{A}_l$ into $\mathcal{S} \times \mathcal{A}$ by defining an equivalent expanded sub-Q function by $Q_l(s, (a^1, \cdots a^l, \cdots, a^L)) := \tilde{Q}_l(s, \tilde{a}^l)$, where the sub-action $a^l = \tilde{a}^l$ and for all $s \in \mathcal{S}, a^j \in \mathcal{A}_j, j \neq l$. We provide an example of the expanded function in Appendix A.1.

Based on this definition, let $\mathbb{Q} := \mathbb{R}^{|\mathcal{S}||\mathcal{A}|\times 1}$, then we define a subspace of $\mathbb{Q}$ associated with $\mathcal{A}_l$ as $\mathbb{Q}_l := \left\{\mathbf{Q} \in \mathbb{Q} \mid \mathbf{Q}_{[s,(a^1,\ldots,a^l,\ldots,a^L)]} = \tilde{\mathbf{Q}}_{l[s,\tilde{a}^l]},\ a^l = \tilde{a}^l\right\}$ for all $s \in \mathcal{S}$ and $a^j \in \mathcal{A}_j, j \neq l$. Note that $\mathbb{Q}_l$ is the set of all vectors whose components are equal if the subscripts take the same value on the sub-action a^l. Finally, we define the factored Q-function space $\hat{\mathbb{Q}}$ as $\hat{\mathbb{Q}} := \left\{\mathbf{Q} \in \mathbb{Q} \;\middle|\; \mathbf{Q} = \sum_{l=1}^{L} \mathbf{Q}_l \text{ where } \mathbf{Q}_l \in \mathbb{Q}_l\right\}$.

Unfortunately, the factored Q-function space is often a subspace of the original Q-function space, implying that the factored Q-value function representation can be restricted. Please see an example in Appendix A.3. This may pose a significant challenge for the convergence analysis of updating the factored Q-function via traditional Bellman optimality operator, as the optimal Q-value function $\mathbf{Q}^\star$ may not be contained in the factored Q-function space.

Therefore, we mimic our proposed factored Q-value update scheme by proposing an approximated factored Bellman optimality operator, which is a composite mapping of a linear projection mapping and the Bellman optimality operator. The operator first updates Q-values via Bellman optimality operator $\mathcal{T}$, and then projects the updated Q-values into the restricted factored Q-function space, which is inspired by approximate dynamic programming (Geramifard et al., 2013; Guestrin et al., 2001). In our algorithm implementation, we actually leverage neural network approximation and gradient update via backpropagation to implicitly perform the composite mapping. To define the projection mapping $\mathbf{G}$, in Appendix A.2 we prove that the factored Q-function space is a *linear* subspace. Then, we define a normalized linear projection mapping onto the closed subspace by $\mathbf{G} := \mathcal{P}_{\hat{\mathbb{Q}}}/\|\mathcal{P}_{\hat{\mathbb{Q}}}\|_\infty$, which is motivated by previous work (Szita & Lorincz, 2008). The projection matrix $\mathcal{P}_{\hat{\mathbb{Q}}}$ is defined by solving an optimization problem $\mathcal{P}_{\hat{\mathbb{Q}}}\mathbf{Q} := \arg\min_{\hat{\mathbf{Q}}\in\hat{\mathbb{Q}}} \left\|\hat{\mathbf{Q}} - \mathbf{Q}\right\|_2$ for any given vector $\mathbf{Q} \in \mathbb{Q}$ (Szita & Lorincz, 2008). Therefore, our approximated factored Bellman optimality operator takes the form of $\hat{\mathcal{T}} := \mathbf{G}\mathcal{T}$. We first show it is a contraction mapping under the norm $\|\cdot\|_\infty$ i.e. $\left\|\hat{\mathcal{T}}[\hat{\mathbf{Q}}_1] - \hat{\mathcal{T}}[\hat{\mathbf{Q}}_2]\right\|_\infty \leq \gamma \left\|\hat{\mathbf{Q}}_1 - \hat{\mathbf{Q}}_2\right\|_\infty$ for any $\hat{\mathbf{Q}}_1, \hat{\mathbf{Q}}_2 \in \hat{\mathbb{Q}}$ in Appendix A.4.

As the operator is a contraction operator on linear subspace $\hat{\mathbb{Q}}$ under norm $\|\cdot\|_\infty$, we can find a unique fixed point $\hat{\mathbf{Q}}^*$ such that $\hat{\mathcal{T}}[\hat{\mathbf{Q}}^*] = \hat{\mathbf{Q}}^*$. Then we show an error bound between the fixed point $\hat{\mathbf{Q}}^*$ and the optimal Q-value function.

Theorem 4.1. *(Error Bound) Let $\mathbf{Q}^*$ be the true optimal Q-value function, then we have*

$$\left\|\hat{\mathbf{Q}}^* - \mathbf{Q}^*\right\|_\infty \leq \frac{\|\mathbf{G}\mathbf{Q}^* - \mathbf{Q}^*\|_\infty}{1-\gamma} \leq \frac{2R}{(1-\gamma)^2} \quad (1)$$

where $\gamma \in [0, 1)$ is the discounted factor, R is the upper bound of rewards.

The Theorem suggests the following. First, the error bound is directly proportional to the projection error of $\mathbf{Q}^*$, which can potentially be small. Second, the upper bound of the error can be constrained to a constant value.

(d) Restore-Then-Explore To promote efficient exploration for high sample efficiency, we propose a restore-then-explore strategy, which is inspired by Ecoffet et al.

(2021). This approach begins by restoring an elite state at the beginning of each episode and subsequently explores the environment using a novelty-seeking exploration bonus. We defer details to Appendix F.4.

4.3. Discussion on HAVE

We first discuss additional advantages of HAVE. Then, we discuss the potential applications of HAVE in addressing other combinatorial optimization (CO) problems. **(1) Advantages** First, HAVE proposes a shared encoder architecture and a data sharing mechanism to share knowledge across tasks, which is a simple yet effective approach to exploit common properties among related tasks. Second, HAVE proposes a restore-then-explore strategy to recycle visited high-performance states, thereby promoting efficient exploration. The idea has been shown successful in hard exploration Atari games (Ecoffet et al., 2021). **(2) Generality** In the hardware design domain, while our experiments focused on the multiplier design task, our approach is applicable to numerous other hardware design problems with similar characteristics. These problems include the design of prefix adders, leading zero detectors, and priority encoders. In the machine learning domain, our proposed hierarchical adaptive MTRL framework provides a novel concept for general multi-objective reinforcement learning. Moreover, the factored Q-value update is applicable to any decision-making problem with a combinatorial action space, which has broad applicability in addressing CO problems.

5. Experiments

Our experiments consist of four main parts. (1) We evaluate HAVE by designing multipliers across a large range of sizes. (2) We perform carefully designed ablation studies to provide further insight into HAVE. (3) We evaluate whether multipliers designed by HAVE can well generalize to large macros widely used in DNN accelerators. (4) We perform visualization experiments and explainability analysis.

Experimental Setup Throughout all experiments, we use the OpenROAD flow (Ajayi & Blaauw, 2019) with NanGate 45nm open-cell library (Nangate Inc., 2008) to synthesize circuits, and use OpenSTA (Parallax Software Inc.) to perform timing analysis, which follows previous work (Zuo et al., 2023). These tools are state-of-the-art open-source EDA tools, and are widely used in research of EDA (Kahng, 2021; Tan et al., 2021; Pilipović et al., 2021). We train our method with SGD (Ruder, 2016) using the PyTorch (Paszke et al., 2019). For fair comparison, we train all methods for 5000 steps in all experiments. Moreover, we set the number of generated preferences each time N as four throughout all experiments. We apply our method to eight multiplier design problems, i.e., designing multipliers with 8-bit, 16-bit, 32-bit, and 64-bit based on AND gate-based and Booth encoding-based PPG, respectively.

Baselines Our baselines include five widely used human-designed, traditional algorithmic, and state-of-the-art (SOTA) learning-based approaches. (1) **Wallace** Tree (Wallace, 1964) is a classical human-designed compression algorithm. (2) **GOMIL** (Xiao et al., 2021) is an algorithmic method based on integer programming. (3) **RL-MUL** (Zuo et al., 2023) is a SOTA RL based method. In addition, we implement two learning-based methods. (4) **Random** uses a random agent to explore the multiplier design space. (5) **RL-Scalar** is the linear scalarized RL-MUL algorithm used in (Roy et al., 2021) instead of using the pareto-driven reward.

Evaluation Metrics We use two evaluation metrics to compare our method with baselines. First, we visualize the approximated Pareto front in terms of the area and delay for multipliers designed by our method and baselines. Second, we use the hypervolume of the approximated Pareto front. We provide details in Appendix C.

Experiment 1. Main Evaluation To demonstrate the superiority of HAVE, we compare HAVE with five competitive baselines on eight multiplier design problems across a wide range of input sizes. The results in Figure 2 demonstrate that multipliers designed by HAVE consistently and significantly Pareto-dominate designs found by all baselines on eight multiplier design problems. In particular, multipliers designed by HAVE achieve a maximum delay reduction of 10% and a maximum area reduction of 7.93% (see Appendix G.2). Moreover, we report the hypervolume of their found Pareto front in Tables 3 and 4 in Appendix G.2. The results demonstrate that HAVE significantly outperforms the baselines in terms of hypervolume, improving the hypervolume by up to 89.05% compared to Wallace, and 28.08% compared to the SOTA RL-MUL. Overall, the results demonstrate that HAVE efficiently approximates the Pareto front, achieving a significant reduction in both area and delay of multipliers.

Experiment 2. Ablation Study To understand the contribution of main components in HAVE, we perform an ablation study on multiplier design problems with 8-bit, 16-bit and 32-bit input widths. The path from RL-MUL (Zuo et al., 2023) to HAVE comprises four main components: restore-then-explore strategy (**R**), factored Q-value update (**F**), multi-task learning (**M**), and hierarchical meta-agent (**H**). To provide insight into the effects behind these components, we build three methods, i.e., the intermediate steps on the incremental path from RL-MUL to HAVE. (1) **R** is RL-MUL with the restore-then-explore strategy for efficient exploration. (2) **FR** is R with the factored Q-function learning. (3) **MFR** is FR with a *static* multi-task learning method. (4) **HAVE=HMFR** is MFR with the hierarchical meta-agent. The results in Table 1 demonstrate that each component in HAVE plays an important role in improving the hypervolume of found Pareto front, especially on multiplier design problems with large input widths. First, **R** outperforms

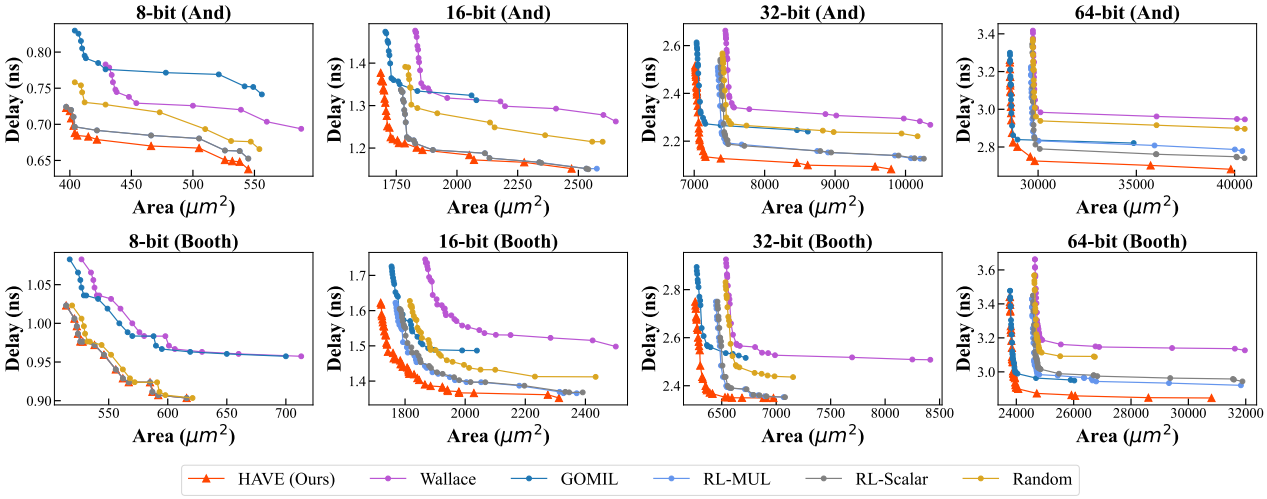


Figure 2. The results demonstrate that multipliers designed by HAVE consistently and significantly Pareto-dominate designs found by all five baselines on eight multiplier design problems.

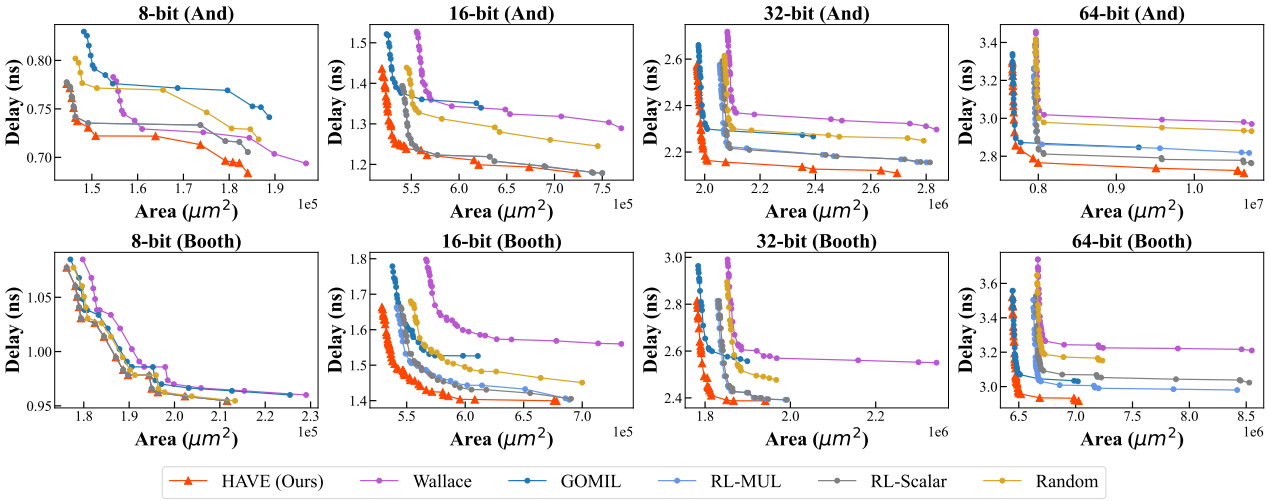


Figure 3. The results demonstrate that PE arrays designed by HAVE consistently and significantly Pareto-dominate designs found by all five baselines on eight multiplier design problems.

RL-MUL on multiplier design problems with 32-bit input widths, demonstrating the importance of promoting efficient exploration on large multipliers. Second, **FR** significantly outperforms **R**, demonstrating the superiority of the factored Q-value update method. Third, **MFR** further improves **FR**, suggesting that exploiting common properties among multiplier optimization problems with diverse preferences is important. Finally, HAVE outperforms **MFR**, demonstrating that the hierarchical framework is important for efficient exploration of both preference and design space.

Experiment 3. Generalization to Large Circuits To evaluate whether multiplier units designed by HAVE can well generalize to large-scale real-world computing circuits with many multiplier units, we deploy multipliers designed by HAVE and baselines into Processing Element (PE) arrays (Park & Chung, 2020; Son et al., 2023), which follows previous work (Zuo et al., 2023). The PE array contains a large number of multipliers, and is widely used in parallel computing tasks and large-scale data processing, such as Deep Neural Network (DNN) accelerators. The results in Figure 3 demonstrate that the PE array with multipliers designed by HAVE consistently and significantly Pareto-dominate PE arrays with multipliers found by baselines. In particular, PE arrays designed by HAVE achieve a maximum delay reduc-

	16-bit Booth		32-bit Booth	
Method	Hypervolume↑	Improvement(%)↑	Hypervolume↑	Improvement(%)↑
Wallace	220.08	NA	1646.54	NA
RL-MUL	376.05	70.87	2272.40	38.01
R	373.50	69.71	2329.80	41.50
FR	397.32	80.53	2395.31	45.48
MFR	410.34	86.45	2409.01	46.31
HAVE (Ours)	**416.07**	**89.05**	**2414.17**	**46.62**

Table 1. The results demonstrate that each component in HAVE plays an important role in improving area and delay of multipliers.

	8-bit Booth		16-bit Booth	
Metrics	Wallace	HAVE	Wallace	HAVE
3:2 Num	29	26	121	119
2:2 Num	13	17	64	15
Total Num	42	43	185	134
Stage Num	3	4	5	4
Minimum Delay	0.9652	0.9117	1.4980	1.3593
Minimum Area	527	521	1867	1723

Table 2. We provide statistics for two typical compressor tree cases, including the number of 3:2 and 2:2 compressors, the minimum area, and the minimum delay.

tion of 10.26% and a maximum area reduction of 6.66% (see Table 8 and Table 9 in Appendix G.4). Moreover, we report the hypervolume of their found Pareto frontiers in Tables 10 and 11 in Appendix G.4. The results demonstrate that HAVE significantly outperforms the baselines in terms of hypervolume, improving the hypervolume by up to 84.79% compared to Wallace, and 28.84% compared to the SOTA RL-MUL. Overall, the results demonstrate that HAVE can well generalize to large-scale computation-intensive circuits, thus significantly reducing the area and delay of real-world computing circuits.

Experiment 4. Visualization and Explainability Analysis To provide further insight into the design mechanism learned by HAVE, we visualize the compressor tree architectures designed by HAVE and Wallace. Due to limited space, we provide visualization results on multiplier design problems in Appendix G.5. Moreover, we provide statistic results for explainability analysis in Table 2. These results suggest the following. (1) As shown in Table 2, the compressor trees designed by HAVE reduces the number of compressors and stages on multipliers with 16 bits input width, which improves the area and delay of multipliers. This is in line with the experience of experts, and thus the results suggest that HAVE effectively learns the expert-knowledge. (2) As shown in Table 2, the results reveal that certain compressor trees generated by HAVE exhibit a paradoxical trend: despite comprising a greater number of stages and compressors, they manifest reduced delay and area. This discrepancy highlights the distinction between analytical evaluation metrics and physical synthesis metrics, thereby emphasizing the importance of employing reinforcement learning for optimizing multipliers with synthesis in the loop.

6. Conclusion

In this paper, we address the challenge of efficiently exploring and finding the optimal design of multipliers. We propose a novel **h**ierarchical p**a**reto-dri**v**en adaptiv**e** (HAVE) multi-task reinforcement learning framework. Specially, HAVE is composed of a meta-agent and an adaptive multi-task agent that can efficiently approximate a set of Pareto-optimal designs for the entire multiplier preference space in a single training. As compared to the existing SOTA RL-MUL method, HAVE significantly improves the hypervolume by up to 28.08%. In the future, we will try to extend our framework to more electronic design tasks.

Acknowledgements

This work was supported in part by National Key R&D Program of China under contract 2022ZD0119801, National Nature Science Foundations of China grants U23A20388, 62021001, U19B2026, and U19B2044. This work was supported in part by Huawei as well. We would like to thank all the anonymous reviewers for their insightful comments.

Impact Statement

This paper presents a sample efficient reinforcement learning method to tackle the multi-objective multiplier optimization problem. As the multiplier is widely used in circuit design as a basic computing unit, this work has a broad impact on chip design. There are many potential societal consequences of our work, none which we feel must be specifically highlighted here.

References

Abels, A., Roijers, D., Lenaerts, T., Nowé, A., and Steckelmacher, D. Dynamic weights in multi-objective deep reinforcement learning. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 11–20. PMLR, 09–15 Jun 2019a. URL https://proceedings.mlr.press/v97/abels19a.html.

Abels, A., Roijers, D., Lenaerts, T., Nowé, A., and Steckelmacher, D. Dynamic weights in multi-objective deep reinforcement learning. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 11–20. PMLR, 09–15 Jun 2019b. URL https://proceedings.mlr.press/v97/abels19a.html.

Agarwal, A., Jiang, N., Kakade, S. M., and Sun, W. Reinforcement learning: Theory and algorithms. *CS Dept., UW Seattle, Seattle, WA, USA, Tech. Rep*, 32, 2019.

Ajayi, T. and Blaauw, D. Openroad: Toward a self-driving, open-source digital layout implementation tool chain. In *Proceedings of Government Microcircuit Applications and Critical Technology Conference*, 2019.

Basaklar, T., Gumussoy, S., and Ogras, U. PD-MORL: Preference-driven multi-objective reinforcement learning algorithm. In *The Eleventh International Conference*

on Learning Representations, 2023. URL https://openreview.net/forum?id=zS9sRyaPFlJ.

Bengio, Y., Lodi, A., and Prouvost, A. Machine learning for combinatorial optimization: a methodological tour d'horizon. *European Journal of Operational Research*, 290(2):405–421, 2021.

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. Exploration by random network distillation. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=H1lJJnR5Ym.

Chen, R., Liu, X.-H., Liu, T.-S., Jiang, S., Xu, F., and Yu, Y. Foresight distribution adjustment for off-policy reinforcement learning. In *Proceedings of the 23rd International Conference on Autonomous Agents and Multiagent Systems*, pp. 317–325, 2024a.

Chen, S., Li, S., Zhuang, Z., Zheng, S., Liang, Z., Ho, T.-Y., Yu, B., and Sangiovanni-Vincentelli, A. L. Floorplet: Performance-aware floorplan framework for chiplet integration. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2023.

Chen, S., Zheng, S., Bai, C., Zhao, W., Yin, S., Bai, Y., and Yu, B. Soc-tuner: An importance-guided exploration framework for dnn-targeting soc design. In *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 207–212. IEEE, 2024b.

Dadda, L. Some schemes for fast serial input multipliers. In *1983 IEEE 6th Symposium on Computer Arithmetic (ARITH)*, pp. 52–59. IEEE, 1983.

Das, B., Paul, A. K., and De, D. An unconventional arithmetic logic unit design and computing in actin quantum cellular automata. *Microsystem Technologies*, pp. 1–14, 2019.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. BERT: pre-training of deep bidirectional transformers for language understanding. In Burstein, J., Doran, C., and Solorio, T. (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pp. 4171–4186. Association for Computational Linguistics, 2019. doi: 10.18653/V1/N19-1423. URL https://doi.org/10.18653/v1/n19-1423.

Dosovitskiy, A., Beyer, L., Kolesnikov, A., Weissenborn, D., Zhai, X., Unterthiner, T., Dehghani, M., Minderer, M., Heigold, G., Gelly, S., Uszkoreit, J., and Houlsby, N. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

Ecoffet, A., Huizinga, J., Lehman, J., Stanley, K. O., and Clune, J. First return, then explore. *Nature*, 590(7847): 580–586, 2021.

Elguibaly, F. A fast parallel multiplier-accumulator using the modified booth algorithm. *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, 47 (9):902–908, 2000.

Gasse, M., Chetelat, D., Ferroni, N., Charlin, L., and Lodi, A. Exact combinatorial optimization with graph convolutional neural networks. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Geng, Z., Li, X., Wang, J., Li, X., Zhang, Y., and Wu, F. A deep instance generative framework for milp solvers under limited data availability. In Oh, A., Naumann, T., Globerson, A., Saenko, K., Hardt, M., and Levine, S. (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 26025–26047. Curran Associates, Inc., 2023.

Geng, Z., Wang, J., Liu, Z., Xu, S., Tang, Z., Yuan, M., Hao, J., Zhang, Y., and Wu, F. Reinforcement learning within tree search for fast macro placement. In *International Conference on Machine Learning*. PMLR, 2024.

Geramifard, A., Walsh, T. J., Tellex, S., Chowdhary, G., Roy, N., How, J. P., et al. A tutorial on linear function approximators for dynamic programming and reinforcement learning. *Foundations and Trends® in Machine Learning*, 6(4):375–451, 2013.

Google DeepMind. Optimizing Computer Systems with More Generalized AI Tools, 2023.

Guestrin, C., Koller, D., and Parr, R. Max-norm projections for factored mdps. In *IJCAI*, volume 1, pp. 673–682, 2001.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In *International conference on machine learning*, pp. 1861–1870. PMLR, 2018a.

Haarnoja, T., Zhou, A., Abbeel, P., and Levine, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of*

Machine Learning Research, pp. 1861–1870. PMLR, 10–15 Jul 2018b. URL https://proceedings.mlr.press/v80/haarnoja18b.html.

Hashemian, R. A new multiplier using wallace structure and carry select adder with pipelining. In *ISCAS '02 Conference Proceedings*, 2002.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016. doi: 10.1109/CVPR.2016.90.

Hillar, C. J. and Lim, L.-H. Most tensor problems are np-hard. *Journal of the ACM (JACM)*, 60(6):1–39, 2013.

Holdsworth, B. *Digital logic design, 2nd ed.* 1987. ISBN 0408015667.

Hu, Y.-Q., Liu, X.-H., Li, S.-Q., and Yu, Y. Cascaded algorithm selection with extreme-region ucb bandit. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(10):6782–6794, 2021.

Huang, G., Hu, J., He, Y., Liu, J., Ma, M., Shen, Z., Wu, J., Xu, Y., Zhang, H., Zhong, K., et al. Machine learning for electronic design automation: A survey. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 26(5):1–46, 2021.

Itoh, N., Tsukamoto, Y., Shibagaki, T., Nii, K., Takata, H., and Makino, H. A 32/spl times/24-bit multiplier-accumulator with advanced rectangular styled wallace-tree structure. In *2005 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 73–76. IEEE, 2005.

Janner, M., Fu, J., Zhang, M., and Levine, S. When to trust your model: Model-based policy optimization. *Advances in neural information processing systems*, 32, 2019.

Jia, C., Zhang, F., Li, Y.-C., Gao, C.-X., Liu, X.-H., Yuan, L., Zhang, Z., and Yu, Y. Disentangling policy from offline task representation learning via adversarial data augmentation. *arXiv preprint arXiv:2403.07261*, 2024.

Kahng, A. B. Advancing placement. In *Proceedings of the 2021 International Symposium on Physical Design*, pp. 15–22, 2021.

Lai, Y., Mu, Y., and Luo, P. Maskplace: Fast chip placement via reinforced visual representation learning. *Advances in Neural Information Processing Systems*, 35:24019–24030, 2022.

Lai, Y., Liu, J., Tang, Z., Wang, B., Hao, J., and Luo, P. Chipformer: Transferable chip placement via offline decision transformer. In *International Conference on Machine Learning*, pp. 18346–18364. PMLR, 2023.

Li, X., Chen, L., Zhang, J., Wen, S., Sheng, W., Huang, Y., and Yuan, M. Effisyn: Efficient logic synthesis with dynamic scoring and pruning. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pp. 1–9, 2023. doi: 10.1109/ICCAD57390.2023.10323902.

Li, X., Li, X., Chen, L., Zhang, X., Yuan, M., and Wang, J. Circuit transformer: End-to-end circuit design by predicting the next gate, 2024a.

Li, X., Zhu, F., Zhen, H.-L., Luo, W., Lu, M., Huang, Y., Fan, Z., Zhou, Z., Kuang, Y., Wang, Z., Geng, Z., Li, Y., Liu, H., An, Z., Yang, M., Li, J., Wang, J., Yan, J., Sun, D., Zhong, T., Zhang, Y., Zeng, J., Yuan, M., Hao, J., Yao, J., and Mao, K. Machine learning insides optverse ai solver: Design principles and applications, 2024b.

Ling, H., Wang, Z., and Wang, J. Learning to stop cut generation for efficient mixed-integer linear programming. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(18):20759–20767, Mar. 2024. doi: 10.1609/aaai.v38i18.30064. URL https://ojs.aaai.org/index.php/AAAI/article/view/30064.

Liu, J., Zhou, S., Zhu, H., and Cheng, C.-K. An algorithmic approach for generic parallel adders. In *ICCAD-2003. International Conference on Computer Aided Design (IEEE Cat. No. 03CH37486)*, pp. 734–740. IEEE, 2003.

Liu, J., Rizzi, C., and Josipović, L. Load-store queue sizing for efficient dataflow circuits. In *2022 International Conference on Field-Programmable Technology (ICFPT)*, pp. 1–9, 2022. doi: 10.1109/ICFPT56656.2022.9974425.

Liu, J., Ni, L., Li, X., Zhou, M., Chen, L., Li, X., Zhao, Q., and Ma, S. Aimap: Learning to improve technology mapping for asics via delay prediction. In *2023 IEEE 41st International Conference on Computer Design (ICCD)*, pp. 344–347. IEEE, 2023a.

Liu, J., Zhou, M., Ma, S., and Pan, L. Mata*: Combining learnable node matching with a* algorithm for approximate graph edit distance computation. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pp. 1503–1512, 2023b.

Liu, X., Xu, F., Zhang, X., Liu, T., Jiang, S., Chen, R., Zhang, Z., and Yu, Y. How to guide your learner: Imitation learning with active adaptive expert involvement. In *Proceedings of the 2023 International Conference on Autonomous Agents and Multiagent Systems*, pp. 1276–1284, London, UK, 2023c.

Liu, X.-H., Xue, Z., Pang, J., Jiang, S., Xu, F., and Yu, Y. Regret minimization experience replay in off-policy reinforcement learning. *Advances in Neural Information Processing Systems*, 34:17604–17615, 2021.

Liu, Z., Hu, H., Zhang, S., Guo, H., Ke, S., Liu, B., and Wang, Z. Reason for future, act for now: A principled framework for autonomous llm agents with provable sample efficiency. *arXiv preprint arXiv:2309.17382*, 2023d.

Liu, Z., Lu, M., Xiong, W., Zhong, H., Hu, H., Zhang, S., Zheng, S., Yang, Z., and Wang, Z. Maximize to explore: One objective function fusing estimation, planning, and exploration. *Advances in Neural Information Processing Systems*, 36, 2024.

Mannor, S. and Shimkin, N. The steering approach for multi-criteria reinforcement learning. In Dietterich, T., Becker, S., and Ghahramani, Z. (eds.), *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2001. URL https://proceedings.neurips.cc/paper_files/paper/2001/file/8c249675aea6c3cbd91661bbae767ff1-Paper.pdf.

Mirhoseini, A., Goldie, A., Yazgan, M., Jiang, J. W., Songhori, E., Wang, S., Lee, Y.-J., Johnson, E., Pathak, O., Nazi, A., et al. A graph placement methodology for fast chip design. *Nature*, 594(7862):207–212, 2021.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. Human-level control through deep reinforcement learning. *Nature*, 518 (7540):529–533, February 2015. ISSN 00280836. URL http://dx.doi.org/10.1038/nature14236.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K. Asynchronous methods for deep reinforcement learning. In Balcan, M. F. and Weinberger, K. Q. (eds.), *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 1928–1937, New York, New York, USA, 20–22 Jun 2016. PMLR. URL https://proceedings.mlr.press/v48/mniha16.html.

Nangate Inc. "Open Cell Library v2008 10 SP1". http://www.nangate.com/openlibrary/, 2008.

Neto, W. L., Moreira, M. T., Amaru, L., Yu, C., and Gaillardon, P.-E. Read your circuit: leveraging word embedding to guide logic optimization. In *Proceedings of the 26th Asia and South Pacific Design Automation Conference*, pp. 530–535, 2021.

Parallax Software Inc. OpenSTA. https://github.com/The-OpenROAD-Project/OpenSTA.

Park, S.-S. and Chung, K.-S. Cenna: cost-effective neural network accelerator. *Electronics*, 9(1):134, 2020.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf.

Pilipović, R., Bulić, P., and Lotrič, U. A two-stage operand trimming approximate logarithmic multiplier. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 68(6):2535–2545, 2021.

Ren, H. and Hu, J. *Machine Learning Applications in Electronic Design Automation*. Springer Nature, 2023.

Roy, R., Raiman, J., Kant, N., Elkin, I., Kirby, R., Siu, M., Oberman, S., Godil, S., and Catanzaro, B. Prefixrl: Optimization of parallel prefix circuits using deep reinforcement learning. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pp. 853–858. IEEE, 2021.

Roy, S., Choudhury, M., Puri, R., and Pan, D. Z. Towards optimal performance-area trade-off in adders by synthesis of parallel prefix structures. In *Proceedings of the 50th Annual Design Automation Conference*, pp. 1–8, 2013.

Ruder, S. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

Sánchez, D., Servadei, L., Kiprit, G. N., Wille, R., and Ecker, W. A comprehensive survey on electronic design automation and graph neural networks: Theory and applications. *ACM Trans. Des. Autom. Electron. Syst.*, 28(2), feb 2023. ISSN 1084-4309. doi: 10.1145/3543853. URL https://doi.org/10.1145/3543853.

Schaul, T., Horgan, D., Gregor, K., and Silver, D. Universal value function approximators. In Bach, F. and Blei, D. (eds.), *Proceedings of the 32nd International Conference on Machine Learning*, volume 37 of *Proceedings of Machine Learning Research*, pp. 1312–1320, Lille, France, 07–09 Jul 2015. PMLR. URL https://proceedings.mlr.press/v37/schaul15.html.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Sharma, S., Suresh, A., Ramesh, R., and Ravindran, B. Learning to factor policies and action-value functions: Factored action space representations for deep reinforcement learning. *arXiv preprint arXiv:1705.07269*, 2017.

Son, H.-W., Al-Hamid, A. A., Na, Y.-S., Lee, D.-Y., and Kim, H.-W. Cnn accelerator using proposed diagonal cyclic array for minimizing memory accesses. *Computers, Materials & Continua*, 76(2), 2023.

Song, J., Roy, R., Raiman, J., Kirby, R., Kant, N., Godil, S., and Catanzaro, B. Multi-objective reinforcement learning with adaptive pareto reset for prefix adder design. In *Workshop on ML for Systems at NeurIPS*, 2022.

Song, J., Swope, A., Kirby, R., Roy, R., Godil, S., Raiman, J., and Catanzaro, B. Circuitvae: Efficient and scalable latent circuit optimization. In *NeurIPS 2023 Workshop on Adaptive Experimental Design and Active Learning in the Real World*, 2023.

Sutton, R. S. and Barto, A. G. *Reinforcement learning: An introduction*. MIT press, 2018.

Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

Szita, I. and Lorincz, A. Factored value iteration converges. *arXiv preprint arXiv:0801.2069*, 2008.

Tan, C., Xie, C., Li, A., Barker, K. J., and Tumeo, A. Aurora: Automated refinement of coarse-grained reconfigurable accelerators. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 1388–1393. IEEE, 2021.

Tesauro, G., Das, R., Chan, H., Kephart, J., Levine, D., Rawson, F., and Lefurgy, C. Managing power consumption and performance of computing systems using reinforcement learning. In Platt, J., Koller, D., Singer, Y., and Roweis, S. (eds.), *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc., 2007. URL https://proceedings.neurips.cc/paper_files/paper/2007/file/c8fbbc86abe8bd6a5eb6a3b4d0411301-Paper.pdf.

Wallace, C. S. A suggestion for a fast multiplier. *IEEE Transactions on electronic Computers*, (1):14–17, 1964.

Wang, C. and Yu, T. Efficient training of multi-task neural solver with multi-armed bandits. *arXiv preprint arXiv:2305.06361*, 2023.

Wang, C., Yang, Y., Han, C., Guo, T., Zhang, H., and Wang, J. A game-theoretic approach for improving generalization ability of tsp solvers. 2021.

Wang, C., Han, C., Guo, T., and Ding, M. Solving uncapacitated p-median problem with reinforcement learning assisted by graph attention networks. *Applied Intelligence*, 53(2):2010–2025, 2023a.

Wang, C., Yu, Z., McAleer, S., Yu, T., and Yang, Y. Asp: Learn a universal neural solver! *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2024a.

Wang, J., Yang, R., Geng, Z., Shi, Z., Ye, M., Zhou, Q., Ji, S., Li, B., Zhang, Y., and Wu, F. Generalization in visual reinforcement learning with the reward sequence distribution. *arXiv preprint arXiv:2302.09601*, 2023b.

Wang, J., Wang, Z., Li, X., Kuang, Y., Shi, Z., Zhu, F., Yuan, M., Zeng, J., Zhang, Y., and Wu, F. Learning to cut via hierarchical sequence/set model for efficient mixed-integer programming, 2024b.

Wang, Z., Wang, J., Zhou, Q., Li, B., and Li, H. Sample-efficient reinforcement learning via conservative model-based actor-critic. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(8): 8612–8620, Jun. 2022. doi: 10.1609/aaai.v36i8.20839. URL https://ojs.aaai.org/index.php/AAAI/article/view/20839.

Wang, Z., Li, X., Wang, J., Kuang, Y., Yuan, M., Zeng, J., Zhang, Y., and Wu, F. Learning cut selection for mixed-integer linear programming via hierarchical sequence model. In *The Eleventh International Conference on Learning Representations*, 2023c. URL https://openreview.net/forum?id=Zob4P9bRNcK.

Wang, Z., Pan, T., Zhou, Q., and Wang, J. Efficient exploration in resource-restricted reinforcement learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 37(8):10279–10287, Jun. 2023d. doi: 10.1609/aaai.v37i8.26224. URL https://ojs.aaai.org/index.php/AAAI/article/view/26224.

Watkins, C. J. and Dayan, P. Q-learning. *Machine learning*, 8:279–292, 1992.

Xiao, W., Qian, W., and Liu, W. Gomil: Global optimization of multiplier by integer linear programming. In *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 374–379. IEEE, 2021.

Yang, R., Sun, X., and Narasimhan, K. A generalized algorithm for multi-objective reinforcement learning and policy adaptation. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019. URL https://proceedings.neurips.cc/paper_files/paper/2019/file/

4a46fbfca3f1465a27b210f4bdfe6ab3-Paper.pdf.

Yang, R., Wang, J., Geng, Z., Ye, M., Ji, S., Li, B., and Wu, F. Learning task-relevant representations for generalization via characteristic functions of reward sequence distributions. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 2242–2252, 2022.

Yao, X., Li, H., Chan, T. H., Xiao, W., Yuan, M., Huang, Y., Chen, L., and Yu, B. Hdldebugger: Streamlining hdl debugging with large language models. *arXiv preprint arXiv:2403.11671*, 2024.

Zhang, J., Gao, Q., Guo, Y., Shi, B., and Luo, G. Easymac: design exploration-enabled multiplier-accumulator generator using a canonical architectural representation. In *2022 27th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 647–653. IEEE, 2022.

Zhu, X., Tang, R., Chen, L., Li, X., Huang, X., Yuan, M., Sheng, W., and Xu, J. A database dependent framework for k-input maximum fanout-free window rewriting. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2023. doi: 10.1109/DAC56929.2023.10247727.

Zuo, D., Ouyang, Y., and Ma, Y. Rl-mul: Multiplier design optimization with deep reinforcement learning. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2023. doi: 10.1109/DAC56929.2023.10247941.

A. Theoretical Analysis

A.1. An Example for Q-Value Function Associated with Sub-Action Space

For a simple example with $L = 2$ and $\mathcal{S} = \left\{s^{(1)}\right\}, \mathcal{A}_1 = \left\{a^{(1,1)}, a^{(1,2)}\right\}$, $\mathcal{A}_2 = \left\{a^{(2,1)}, a^{(2,2)}\right\}$, for any function $Q_1(s, \mathbf{a}) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ associated with $\mathcal{A}_1$, it's vectored form

$$\mathbf{Q}_1 = \begin{bmatrix} Q_1\left(s^{(1)}, \left(a^{(1,1)}, a^{(2,1)}\right)\right) \\ Q_1\left(s^{(1)}, \left(a^{(1,1)}, a^{(2,2)}\right)\right) \\ Q_1\left(s^{(1)}, \left(a^{(1,2)}, a^{(2,1)}\right)\right) \\ Q_1\left(s^{(1)}, \left(a^{(1,2)}, a^{(2,2)}\right)\right) \end{bmatrix}$$

Now since the value of $Q_1(s, \mathbf{a})$ has nothing to do with sub-action $a^2 \in \mathcal{A}_2$, therefore this vector always has the form like $[a, a, b, b]^T$. Similarly, the vector of any function associated with $\mathcal{A}_2$ has the form of $[c, d, c, d]^T$.

A.2. Useful Lemmas

Lemma A.1. *Set $\mathbb{Q}_l$ associated with sub-action space $\mathcal{A}_l$ is defined by*

$$\mathbb{Q}_l := \left\{\mathbf{Q} \in \mathbb{Q} \mid \mathbf{Q}_{[s,(a^1,\ldots,a^l,\ldots,a^L)]} = \tilde{\mathbf{Q}}_{l[s,a^l]}\right\} \tag{2}$$

where sub-action a^l traverses the set $\mathcal{A}_l$ and $\forall a^j \in \mathcal{A}_j, j \neq l, \forall s \in \mathcal{S}$. Then $\mathbb{Q}_l$ is a linear subspace of $\mathbb{Q} := \mathbb{R}^{|\mathcal{S}||\mathcal{A}|\times 1}$.

Proof. From definition of $\mathbb{Q}_l$ we have $\forall \mathbf{Q} \in \mathbb{Q}_l$, then $\mathbf{Q}_{[s,(a^1,\ldots,a^l,\ldots,a^L)]} = \mathbf{Q}_{\left[s,\left(a^{1'},\ldots,a^l,\ldots,a^{L'}\right)\right]}$, $\forall a^i, a^{i'} \in \mathcal{A}_i, i \neq l$ and $a^l \in \mathcal{A}_l$, $s \in \mathcal{S}$.

For a sub-action space $\mathcal{A}_l$, obviously $\mathbf{0} := (0, 0, \ldots, 0)^T$ has zeros in every position and thus $\mathbf{0} \in \mathbb{Q}_l$.

Then for any $\mathbf{Q}_1, \mathbf{Q}_2 \in \mathbb{Q}_l$ we have $\forall a^i, a^{i'} \in \mathcal{A}_i, i \neq l$ and $\forall a^l \in \mathcal{A}_l, s \in \mathcal{S}$

$$\begin{aligned}(\mathbf{Q}_1 + \mathbf{Q}_2)_{[s,(a^1,\ldots,a^l,\ldots,a^L)]} &= \mathbf{Q}_{1[s,(a^1,\ldots,a^l,\ldots,a^L)]} + \mathbf{Q}_{2[s,(a^1,\ldots,a^l,\ldots,a^L)]} \\ &= \mathbf{Q}_{1\left[s,\left(a^{1'},\ldots,a^l,\ldots,a^{L'}\right)\right]} + \mathbf{Q}_{2\left[s,\left(a^{1'},\ldots,a^l,\ldots,a^{L'}\right)\right]} \\ &= (\mathbf{Q}_1 + \mathbf{Q}_2)_{\left[s,\left(a^{1'},\ldots,a^l,\ldots,a^{L'}\right)\right]}\end{aligned}$$

Therefore $\mathbf{Q}_1 + \mathbf{Q}_2 \in \mathbb{Q}_l$. Finally, $\forall \mathbf{Q} \in \mathbb{Q}_l$ and $\lambda \in \mathbb{R}$ we have $\forall a^i, a^{i'} \in \mathcal{A}_i, i \neq l$ and $\forall a^l \in \mathcal{A}_l, s \in \mathcal{S}$

$$\begin{aligned}(\lambda\mathbf{Q})_{[s,(a^1,\ldots,a^l,\ldots,a^L)]} &= \lambda\mathbf{Q}_{[s,(a^1,\ldots,a^l,\ldots,a^L)]} \\ &= \lambda\mathbf{Q}_{\left[s,\left(a^{1'},\ldots,a^l,\ldots,a^{L'}\right)\right]} \\ &= (\lambda\mathbf{Q})_{\left[s,\left(a^{1'},\ldots,a^l,\ldots,a^{L'}\right)\right]}\end{aligned}$$

Therefore $\lambda\mathbf{Q} \in \mathbb{Q}_l$.

Since $\mathbb{Q}$ is a linear space, so set $\mathbb{Q}_l$ is a linear subspace of $\mathbb{Q}$

□

Lemma A.2. *The restricted Q-function space $\hat{\mathbb{Q}}$ defined by*

$$\hat{\mathbb{Q}} := \left\{\mathbf{Q} \in \mathbb{Q} \;\middle|\; \mathbf{Q} = \sum_{l=1}^{L} \mathbf{Q}_l \text{ where } \mathbf{Q}_l \in \mathbb{Q}_l\right\} \tag{3}$$

is a linear subspace of $\mathbb{Q} := \mathbb{R}^{|\mathcal{S}||\mathcal{A}|\times 1}$.

Proof. $\mathbf{0} = \mathbf{0} + \mathbf{0} + \cdots + \mathbf{0}$ and $\mathbf{0} \in \mathbb{Q}_l$ for $l = 1, 2, \ldots, L$, then $\mathbf{0} \in \hat{\mathbb{Q}}$.

Then, for any $\mathbf{Q}^{(1)}, \mathbf{Q}^{(2)} \in \hat{\mathbb{Q}}$, there exists $\mathbf{Q}_l^{(1)} \in \mathbb{Q}_l,\ \mathbf{Q}_l^{(2)} \in \mathbb{Q}_l,\ l = 1, 2, \ldots, L$ such that $\mathbf{Q}^{(1)} = \sum_{l=1}^{L} \mathbf{Q}_l^{(1)}$ and $\mathbf{Q}^{(2)} = \sum_{l=1}^{L} \mathbf{Q}_l^{(2)}$.

Therefore, we have $\mathbf{Q}^{(1)} + \mathbf{Q}^{(2)} = \sum_{l=1}^{L} \left[\mathbf{Q}_l^{(1)} + \mathbf{Q}_l^{(2)}\right]$. Since $\mathbb{Q}_l$ is a linear subspace, therefore $\mathbf{Q}_l^{(1)} + \mathbf{Q}_l^{(2)} \in \mathbb{Q}_l,\ l = 1, 2, \ldots, L$. According to definition, $\mathbf{Q}^{(1)} + \mathbf{Q}^{(2)} \in \hat{\mathbb{Q}}$.

Finally, for any $\mathbf{Q} \in \hat{\mathbb{Q}},\ \lambda \in \mathbb{R}$, there exists $\mathbf{Q}_l \in \mathbb{Q}_l,\ l = 1, 2, \ldots, L$ such that $\lambda\mathbf{Q} = \lambda \sum_{l=1}^{L} \mathbf{Q}_l = \sum_{l=1}^{L} \lambda\mathbf{Q}_l$. Since $\mathbb{Q}_l$ is a linear subspace, therefore $\lambda\mathbf{Q}_l \in \mathbb{Q}_l,\ l = 1, 2, \ldots, L$. Then according to definition, we have $\lambda\mathbf{Q} \in \hat{\mathbb{Q}}$.

Since $\mathbb{Q}$ is a linear space, so set $\hat{\mathbb{Q}}$ is a linear subspace of $\mathbb{Q}$ □

A.3. An example for $\hat{\mathbb{Q}}$ is a proper subspace of $\mathbb{Q}$

There exists cases that $\hat{\mathbb{Q}}$ is a proper subspace of $\mathbb{Q}$, i.e. $\mathbf{dim}\hat{\mathbb{Q}} < \mathbf{dim}\mathbb{Q}$. Let $L = 2$ and $\mathcal{S} = \left\{s^{(1)}\right\}, \mathcal{A}_1 = \left\{a^{(1,1)}, a^{(1,2)}\right\},\ \mathcal{A}_2 = \left\{a^{(2,1)}, a^{(2,2)}\right\}$, for any function $Q(s, \mathbf{a}) : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, it's vectored form

$$\mathbf{Q} = \begin{bmatrix} Q\left(s^{(1)}, \left(a^{(1,1)}, a^{(2,1)}\right)\right) \\ Q\left(s^{(1)}, \left(a^{(1,1)}, a^{(2,2)}\right)\right) \\ Q\left(s^{(1)}, \left(a^{(1,2)}, a^{(2,1)}\right)\right) \\ Q\left(s^{(1)}, \left(a^{(1,2)}, a^{(2,2)}\right)\right) \end{bmatrix}$$

As discussed in Appendix A.1, vectors in $\mathbb{Q}_1$ has the form like $[a, a, b, b]^T$, so we can easily find a set of basis $\mathbf{h}_1 = [1, 1, 0, 0]^T,\ \mathbf{h}_2 = [0, 0, 1, 1]^T$; vectors in $\mathbb{Q}_2$ has the form like $[c, d, c, d]^T$, then a set of basis can be $\mathbf{h}_3 = [1, 0, 1, 0]^T,\ \mathbf{h}_4 = [0, 1, 0, 1]^T$. Now we construct a new matrix $\mathbf{H} := (\mathbf{h}_1,\ \mathbf{h}_2,\ \mathbf{h}_3,\ \mathbf{h}_4)$. According to definition, $\hat{\mathbb{Q}}$ is the column space of $\mathbf{H}$. Now that

$$\mathbf{dim}\hat{\mathbb{Q}} = \mathbf{rank}\mathbf{H} = \mathbf{rank}\begin{bmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{bmatrix} = 3$$

However, the dimension of space $\mathbb{Q} = \mathbb{R}^{4\times 1}$ is 4. Thus $\hat{\mathbb{Q}}$ is a proper subspace of $\mathbb{Q}$. For example vector $[1, 0, 0, 0]^T \notin \hat{\mathbb{Q}}$. Consequently, subsapce $\hat{\mathbb{Q}}$ carries less information than the original space $\mathbb{Q}$.

A.4. Theorem that approximated factored Bellman optimality operator $\hat{\mathcal{T}}$ is a contraction mapping

Theorem A.3. *(Contraction Mapping) Approximated factored Bellman optimality operator $\hat{\mathcal{T}}$ is a contraction mapping under the norm $\|\cdot\|_\infty$, i.e. $\forall \hat{\mathbf{Q}}_1, \hat{\mathbf{Q}}_2 \in \hat{\mathbb{Q}}$,*

$$\left\|\hat{\mathcal{T}}[\hat{\mathbf{Q}}_1] - \hat{\mathcal{T}}[\hat{\mathbf{Q}}_2]\right\|_\infty \leq \gamma \left\|\hat{\mathbf{Q}}_1 - \hat{\mathbf{Q}}_2\right\|_\infty \tag{4}$$

where $\gamma \in [0, 1)$ is the discounted factor.

Proof. It has been proved that Bellman optimality operator $\mathcal{T}$ is a contracted operator in the sense of the infinity norm $\|\cdot\|_\infty$(Agarwal et al., 2019):

$$\|\mathcal{T}[\mathbf{Q}_1] - \mathcal{T}[\mathbf{Q}_2]\|_\infty \leq \gamma \|\mathbf{Q}_1 - \mathbf{Q}_2\|_\infty$$

For any $\hat{\mathbf{Q}}_1, \hat{\mathbf{Q}}_2 \in \hat{\mathbb{Q}}$

$$\begin{aligned}
\left\|\hat{\mathcal{T}}(\hat{\mathbf{Q}}_1) - \hat{\mathcal{T}}(\hat{\mathbf{Q}}_2)\right\|_\infty &= \left\|\mathbf{G}\mathcal{T}(\hat{\mathbf{Q}}_1) - \mathbf{G}\mathcal{T}(\hat{\mathbf{Q}}_2)\right\|_\infty \\
&= \left\|\mathbf{G}[\mathcal{T}(\hat{\mathbf{Q}}_1) - \mathcal{T}(\hat{\mathbf{Q}}_2)]\right\|_\infty \\
&\leq \|\mathbf{G}\|_\infty \left\|\mathcal{T}(\hat{\mathbf{Q}}_1) - \mathcal{T}(\hat{\mathbf{Q}}_2)\right\|_\infty \\
&= \left\|\mathcal{T}(\hat{\mathbf{Q}}_1) - \mathcal{T}(\hat{\mathbf{Q}}_2)\right\|_\infty \\
&\leq \gamma \left\|\hat{\mathbf{Q}}_1 - \hat{\mathbf{Q}}_2\right\|_\infty
\end{aligned}$$

□

A.5. Proof for Theorem 4.1

Assume that scalared reward function is bounded: $|\mathbf{w}^T\mathbf{r}(s, a|d)| \leq R$, then we have the following inequality

$$\left\|\hat{\mathbf{Q}}^* - \mathbf{Q}^*\right\|_\infty \leq \frac{\|\mathbf{G}\mathbf{Q}^* - \mathbf{Q}^*\|_\infty}{1-\gamma} \leq \frac{2R}{(1-\gamma)^2} \tag{5}$$

where $\hat{\mathcal{T}}[\hat{\mathbf{Q}}^*] = \hat{\mathbf{Q}}^*,\ \mathbf{Q}^* = \mathcal{T}[\mathbf{Q}^*]$ are fixed points of either Bellman optimality operators, and $\gamma \in [0, 1)$ is the discounted factor of MDP.

Proof.

$$\begin{aligned}
\left\|\hat{\mathbf{Q}}^* - \mathbf{Q}^*\right\|_\infty &= \left\|\hat{\mathcal{T}}(\hat{\mathbf{Q}}^*) - \mathcal{T}(\mathbf{Q}^*)\right\|_\infty \\
&= \left\|\hat{\mathcal{T}}(\hat{\mathbf{Q}}^*) - \hat{\mathcal{T}}(\mathbf{Q}^*) + \hat{\mathcal{T}}(\mathbf{Q}^*) - \mathcal{T}(\mathbf{Q}^*)\right\|_\infty \\
&\leq \left\|\hat{\mathcal{T}}(\hat{\mathbf{Q}}^*) - \hat{\mathcal{T}}(\mathbf{Q}^*)\right\|_\infty + \left\|\hat{\mathcal{T}}(\mathbf{Q}^*) - \mathcal{T}(\mathbf{Q}^*)\right\|_\infty \\
&= \left\|\mathbf{G}\mathcal{T}(\hat{\mathbf{Q}}^*) - \mathbf{G}\mathcal{T}(\mathbf{Q}^*)\right\|_\infty + \|\mathbf{G}\mathcal{T}(\mathbf{Q}^*) - \mathcal{T}(\mathbf{Q}^*)\|_\infty \\
&= \left\|\mathbf{G}\left[\mathcal{T}(\hat{\mathbf{Q}}^*) - \mathcal{T}(\mathbf{Q}^*)\right]\right\|_\infty + \|\mathbf{G}\mathbf{Q}^* - \mathbf{Q}^*\|_\infty \\
&\leq \|\mathbf{G}\|_\infty \left\|\mathcal{T}(\hat{\mathbf{Q}}^*) - \mathcal{T}(\mathbf{Q}^*)\right\|_\infty + \|\mathbf{G}\mathbf{Q}^* - \mathbf{Q}^*\|_\infty \\
&= \left\|\mathcal{T}(\hat{\mathbf{Q}}^*) - \mathcal{T}(\mathbf{Q}^*)\right\|_\infty + \|\mathbf{G}\mathbf{Q}^* - \mathbf{Q}^*\|_\infty \\
&\leq \gamma \left\|\hat{\mathbf{Q}}^* - \mathbf{Q}^*\right\|_\infty + \|\mathbf{G}\mathbf{Q}^* - \mathbf{Q}^*\|_\infty
\end{aligned}$$

After rearranging, we obtain that

$$\left\|\hat{\mathbf{Q}}^* - \mathbf{Q}^*\right\|_\infty \leq \frac{\|\mathbf{G}\mathbf{Q}^* - \mathbf{Q}^*\|_\infty}{1-\gamma} \leq \frac{\|\mathbf{G} - I\|_\infty \|\mathbf{Q}^*\|_\infty}{1-\gamma} \leq \frac{2}{1-\gamma}\|\mathbf{Q}^*\|_\infty$$

Denote π^* as the optimal policy, $(s_m, \mathbf{a}_m) := \arg\max_{(s,\mathbf{a})\in\mathcal{S}\times\mathcal{A}} |\mathbf{Q}^*_{(s,\mathbf{a})}|$. Then according to definition of action value function Q^*:

$$\begin{aligned}
\|\mathbf{Q}^*\|_\infty &= |Q^*(s_m, \mathbf{a}_m|\mathbf{w}, d)| \\
&= \left| \mathbb{E}_{\pi^*, f} \left[\sum_{t=0}^{\infty} \gamma^t \mathbf{w}^T \mathbf{r}(s_t, \mathbf{a}_t|d) \middle| s_0 = s_m, \mathbf{a}_0 = a_m \right] \right| \\
&\leq \mathbb{E}_{\pi^*, f} \left[\left| \sum_{t=0}^{\infty} \gamma^t \mathbf{w}^T \mathbf{r}(s_t, \mathbf{a}_t|d) \right| \middle| s_0 = s_m, \mathbf{a}_0 = \mathbf{a}_m \right] \\
&\leq \mathbb{E}_{\pi^*, f} \left[\sum_{t=0}^{\infty} \left| \gamma^t \mathbf{w}^T \mathbf{r}(s_t, \mathbf{a}_t|d) \right| \middle| s_0 = s_m, \mathbf{a}_0 = \mathbf{a}_m \right] \\
&\leq \mathbb{E}_{\pi^*, f} \left[\sum_{t=0}^{\infty} \left| \gamma^t R \right| \middle| s_0 = s_m, \mathbf{a}_0 = \mathbf{a}_m \right] \\
&= \frac{R}{1-\gamma}
\end{aligned}$$

where $s_{t+1} \sim f(\cdot|s_t, \mathbf{a}_t)$ and $\mathbf{a}_t \sim \pi^*(\cdot|s_t)$. Consequently, we have the maximum approximation error is constrained by:

$$\left\| \hat{\mathbf{Q}}^* - \mathbf{Q}^* \right\|_\infty \leq \frac{2R}{(1-\gamma)^2}$$

□

B. Additional Related Work

Multi-Objective Reinforcement Learning Reinforcement learning can be generally divided into model-free (Haarnoja et al., 2018a; Wang et al., 2023d; Yang et al., 2022; Liu et al., 2024; Wang et al., 2023b; Liu et al., 2021), model-based (Janner et al., 2019; Liu et al., 2023d; Wang et al., 2022), and offline RL (Hu et al., 2021; Chen et al., 2024a; Jia et al., 2024; Liu et al., 2023c) approaches. In this paper, our HAVE falls into the model-free category. Roughly speaking, MORL methods fall into two categories: single-policy and multi-policy approaches. Single-policy approaches (Mannor & Shimkin, 2001; Tesauro et al., 2007) transform a multi-objective problem into a single-objective problem by scalarizing the rewards based on a given preference between objectives. However, the multiplier optimization problem is a preference-agnostic problem, and they struggle to find Pareto-optimal circuit designs when preferences are unknown.

Multiplier Circuit Design In general, the approaches for multiplier circuit design fall into three categories as follows. First, manual designs (Wallace, 1964; Dadda, 1983; Itoh et al., 2005) are obtained by leveraging human expertise architectures from regular architectures which require high engineering effort. Second, traditional algorithmic methods (Xiao et al., 2021; Liu et al., 2003; Roy et al., 2013) generate circuit architectures based on particular strategies, such as mathematical programming and heuristic search. However, they rely on proxy metrics to optimize circuits, such as the size and/or depth of a circuit, which results in a gap with real end flow. Third, recent methods (Zuo et al., 2023; Roy et al., 2021; Song et al., 2022) propose to use reinforcement learning to optimize circuits based on the actual evaluation metrics within the optimization loop, which offers promising approaches to bridge the gap of proxy metrics. However, the existing methods suffer from inefficient exploration and thus tend to find sub-optimal designs due to the large combinatorial search space and the intricate balance of multiple conflicting objectives. Moreover, recent work (Song et al., 2023) proposes to use generative models to generate circuit designs.

Machine Learning for Chip Design As chip complexity has grown exponentially with the development of semiconductor technology, using machine learning (ML) to assist the automated chip design workflow has been an active topic of significant interest in recent years (Mirhoseini et al., 2021; Huang et al., 2021; Sánchez et al., 2023; Neto et al., 2021; Lai et al., 2022; 2023). The chip design workflow consists of many stages (Huang et al., 2021; Ren & Hu, 2023), such as high-level synthesis (Yao et al., 2024; Liu et al., 2022), logic synthesis (Li et al., 2023; Zhu et al., 2023; Li et al., 2024a; Liu et al., 2023a;b), placement (Lai et al., 2022; 2023; Geng et al., 2024; Chen et al., 2023), design space exploration (Chen et al., 2024b), etc.

Machine Learning for Combinatorial Optimization Optimizing multiplier circuit designs is also essentially a combinatorial optimization problem. The use of machine learning to tackle combinatorial optimization problems has been an active

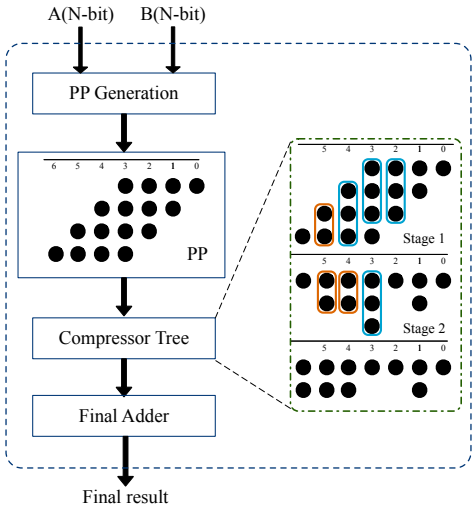


Figure 4. Multiplier Architecture Details

topic of significant interest in recent years (Bengio et al., 2021; Gasse et al., 2019; Geng et al., 2023; Wang et al., 2023c; 2024b; Ling et al., 2024; Li et al., 2024b; Wang et al., 2021; 2024a; 2023a; Wang & Yu, 2023).

C. More Details of Background

As illustrated in Figure 4, the architecture of the multiplier begins with the Partial Product Generator (PPG), which generates partial products based on the multiplicand and multiplier. The PPG is comprised of two primary types: one based on AND gates, as demonstrated by the partial products generated in Figure 4, and the other on Booth encoding. For an $M_b \times N_b$-bit multiplier, the AND gate-based PPG uses $M_b \times N_b$ AND gates to produce an N_b row $(M_b + N_b)$ column bit matrix. Similarly, the Booth encoding-based PPG with a radix of r reduces the number of partial products by making a group of r bits of the multiplier and produces $\lceil (N_b + 1)/r \rceil$ rows of $(M_b + 1)$ bits. Subsequently, the Compressor Tree (CT) utilizes compressors in multiple stages to parallelly compress the partial products down to only two rows. Therefore, only the compressor tree structures that leave either one or two partial products in each column by the final stage are considered legal structures. The CPA is also a key arithmetic circuit used to sum up the two rows and yield the final result. In the optimization of CPA, we often map it to a prefix computation problem using intermediate generate (G) signals and propagate (P) signals. In this way, we can parallelly generate the final results, breaking the dependency of serial carry propagation.

RL-MUL uses $\mathcal{T} \in \mathbf{R}^{K\times(M_b+N_b)\times ST}$ as the state presentation, where K is the total kinds of compressors, ST is the compress stage number and N_b, M_b are the input width. For an element t_{ij}^k in $\mathcal{T}$, it indicates the i-th kind of compressors is used at column j stage i. Given a matrix $M \in \mathbf{R}^{K\times(N_b+M_b)}$ where the element m_{ij} denotes the number of i-th compressor used in column j, RL-MUL gets the $\mathcal{T}$ by assign the overall number of compressors into different stages following the designed scheme. When selecting the action, RL-MUL only considers the legal action that makes the operation at existing compressors and leads the final production products to 1 or 2. Moreover, the action at column j will influence the column (j+1) as the carry bit. RL-MUL uses a legalization strategy to refine the state from the column (j+1) to the most significant bit to ensure the PPs of every line to 1 or 2 after actions.

For an n-objective optimization problem, a solution x Pareto dominates another solution y if x is not worse than y in all objectives and has at least one strictly better value, i.e., $\forall i \in [1, n], f_i(x) \leq f_i(y) \wedge \exists i \in [1, n], f_i(x) < f_i(y)$. A Pareto optimal solution is one that is not dominated by any solution, and the set composed of all Pareto optimal solutions is referred to as the Pareto optimal solution set. The projection of the Pareto optimal solution set in the objective space is called the Pareto front. The hypervolume of the Pareto front is the volume of the region within the boundaries defined by reference points. When reference points are fixed, a Pareto solution set with a larger hypervolume is considered superior.

Definition C.1 (Hypervolume metric)**.** Let P be a Pareto front approximation in an m-dimensional objective space and

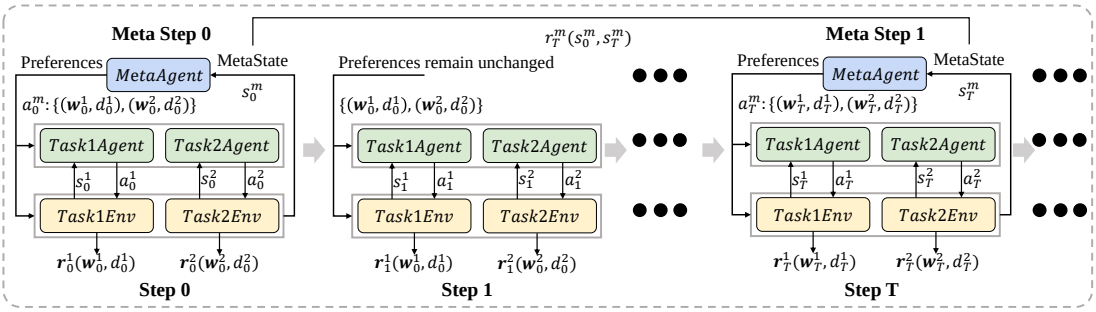


Figure 5. Overview of the hierarchical framework in HAVE with two generated preferences, i.e., $N = 2$.

contains N solutions. Let $r_0 \in R^m$ be the reference point. Then, the hypervolume metric is defined as:

$$\mathcal{H}(P, r_0) = \int_{R^m} \mathbb{1}_{H(P,r_0)(z)dz}$$

, where $H(P, r_0) = \{z \in Z | \exists 1 \leq i \leq |P| : r_0 \preceq z \preceq P(i)\}$.$P(i)$ is the i-th solution in P, $\preceq$ is the relation operator of objective dominance, and $\mathbb{1}_{H(P,r_0)}$is a Dirac delta function that equals 1 if $z \in H(P, r_0)$ and 0 otherwise.

D. Implementation Details of the Baselines

GOMIL (Xiao et al., 2021) is a global optimization method that simultaneously considers the CT and CPA. The author provides the open-source C++ code. We can extract the required structure from its solution files.

RL-MUL (Zuo et al., 2023) encodes the state into a tensor $\mathcal{T}$ described in Section 3 and Appendix C, using ResNet-18 as the network backbone and training based on the DQN algorithm. Different from the Random method, RL-MUL only chooses the action randomly in warm-up steps. In future steps, it chooses the action that can maximize the masked Q-value of the network.

RL-Scalar uses the method called the scalarized deep Q-learning algorithm (Roy et al., 2021), which scalarizes the Q values with a weight vector ω.

We encode our CT following EasyMAC (Zhang et al., 2022) rules and use it to generate Verilog files. In EasyMAC, a legal CT can be represented as a sequence $s_{ct} = p_0 p_1 \cdots p_r$, where each $p_i = (index_i, type_i)$ signifies the index and type of a compressor. When converting CT into a sequence, we need to follow the order: from a higher column to a lower column, from the earlier stage to the next stage, and from a compressor with a larger encoded type to a lower one. Moreover, to ensure a fair comparison, we uniformly use the default adder provided by the synthesis tool for our CPA implementations.

E. Implementation Details on the Meta Agent

E.1. Overview of HAVE

We provide an overview of HAVE in Figure 5.

E.2. Discussion on the Challenge of Delay Constraints Augmented Preference Space

First, the augmented preference space is much larger than the original preference space due to its combinatorial structure between weight vectors and delay constraints. Second, in general MORL tasks, the reward function of the environment is invariant across diverse preferences. In contrast, the reward function significantly varies across different preferences in our augmented preference space, as the delay constraints significantly impact the reward of the circuit synthesis environment. Thus, the existing offline preference vector random sampling strategy (Yang et al., 2019; Basaklar et al., 2023) suffers from inefficient exploration, and can be inapplicable to our problem.

E.3. Designed Meta State Features

To identify which preference region is under-explored, we design the following meta-state features based on Pareto points distribution. To approximate the Pareto points distribution, we track the evolving Pareto front, and divide the interval

between the maximum area and minimum area N equally, where N is equal to the number of decomposed preference spaces. Similarly, we divide the interval between the maximum delay and minimum delay N equally as well. Therefore, we propose to count the number of Pareto points in each sub-interval to approximate the Pareto points distribution. As a result, we obtain a $2N$-dimensional feature vector. The scarcity of Pareto points necessitates a more frequent exploration of its corresponding preference space.

E.4. Meta Policy Training

Policy Network Let π_m denote the meta policy $\pi_m : \mathcal{S}_m \to \mathcal{P}(\mathcal{A}_m)$, where $\mathcal{P}(\mathcal{A}_m)$ denotes the probability distribution over the meta action space, and $\pi_m(\cdot|s)$ denotes the probability distribution over the meta action space given the state s. Note that the meta action space is a joint action space, and thus the probability $\pi_m(a_m|s_m)$ denotes the joint probability of actions $\pi_m([a_m^1, \ldots, a_m^N]|s_m)$, where $a_m = [a_m^1, \ldots, a_m^N]$, $a_m \in \mathcal{A}_m$, and $a_m^i \in \mathcal{A}_m^i$, $i = 1, \ldots N$. For simplicity and sample-efficiency, we assume that the sub-actions $a_m^1, \ldots, a_m^N$ are conditionally independent, which is commonly used in deep reinforcement learning (Schulman et al., 2017; Haarnoja et al., 2018b). Therefore, we get $\pi_m(a_m|s_m) = \Pi_{i=1}^N \pi_m^i(a_m^i|s_m)$. To parameterize the meta-policy, we propose to use a multi-head neural network, where each head π_{θ_i} approximates the probability distribution $\pi_m^i(\cdot|s_m)$. The multi-head neural network is a shared neural network architecture with N heads branching off independently. To train the meta policy, we use a reinforce algorithm following the well-known policy gradient theorem (Sutton et al., 1999; Sutton & Barto, 2018). We provide implementation details in Appendix E.

Due to the high cost of collecting rewards in the multiplier optimization environment and the data used to train the meta-agent being acquired over an extended time scale, our meta-agent must achieve high sample efficiency to acquire an effective preference generation strategy with limited data. Consequently, we implement two simplifications in the algorithm. Initially, we simplify the action space. Due to the substantial scale difference between the weight vector and delay constraints, as well as a consistent relationship existing between delay constraints and the preference for delay, we model the preference generation problem as a "learn to refine" problem, initiating with a key preference and iteratively refining it. Specifically, we reduce the action space for each sub-preference space to two actions: 0 and 1. Action 0 signifies an increase in the preference for delay, whereas action 1 indicates a decrease in the preference for delay. Secondly, we simplify the problem into a single-step contextual bandit problem, with each episode having a length of 1.

Based on the above simplified problem modeling, we use softmax distribution to model the policy on each sub-action space $\pi_m^i(a_m^i|s_m)$. Specifically, we use a multi-head neural network to parameterize the joint policy $\pi_{\theta_m}(a^m|s^m)$, where the i-th head models the softmax distribution $\pi_{\theta_m^i}(a_m^i|s_m)$. The multi-head neural network contains two hidden layers with 128 units and the ReLu activation function. Finally, based on the above parameterization policy, we use the reinforce algorithm for training. We use Adam as the optimizer and use a learning rate of 1e-3.

F. Implementation Details on the Adaptive Multi-Task Agent

F.1. Hardware Specification

Our experiments were executed on a Linux-based system equipped with a 3.60 GHz Intel Xeon Gold 6246R CPU and NVIDIA RTX 3090 GPU.

F.2. Synthesis Tool Setup

Nangate45 is a widely used standard cell library in the semiconductor industry. It is open source and free, and we can obtain it at `https://silvaco.com/services/library-design/`. Readers can refer to `https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts`, seeking the artifact of OpenROAD flow matched with the distribution. EasyMAC is a MAC generator based on Chisel, which takes a sequence representation encoded by its rules as input to generate the Verilog code. We can directly download the code at `https://github.com/pku-dasys/easymac`. One thing to note is that the code can only generate the MAC based on AND-Gate. So we implement the code to gain the multiplier based on booth-encoding.

F.3. Q-Network Architecture

We use the ResNet-18 as the image encoder, and use a non-parametric normalize layer as the preference encoder. In terms of the preference encoder, We normalize the delay constraints to the range [0,1] by dividing by the maximum delay constraint.

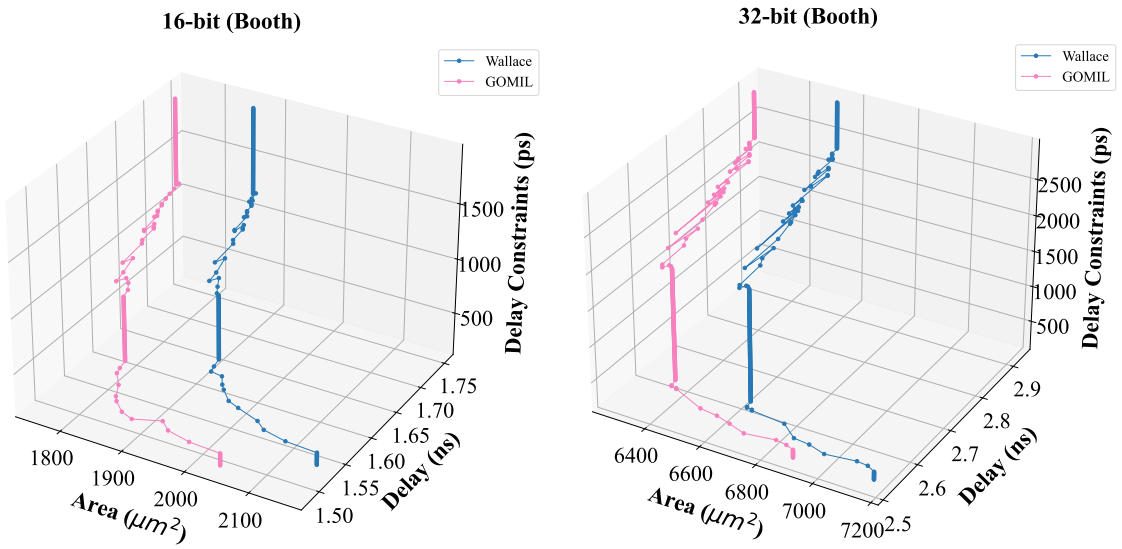


Figure 6. We visualize the area-delay curves for a multiplier designed by Wallace (Wallace, 1964) or GOMIL (Xiao et al., 2021) with increasing synthesis-related delay constraints from a 3D perspective

Then, our multi-head decoder is comprised of multiple multi-layer perceptrons (MLPs). Each MLP contains two hidden layers with 256 units and the ReLu activation function. To train the Q-network asynchronously, we use an individual SGD optimizer for each thread, and set the learning rate as 1e-2.

F.4. Details on the Restore then Explore

Specifically, we formulate the environment as a restorable environment, and maintain an elite pool to store visited states with high area/delay. At the beginning of each episode, we restore to a state sampled from this elite pool. Then, we propose to encourage the agent to explore unvisited states by a novelty-seeking exploration bonus. For simplicity in implementation, we instantiate the restoring strategy as a randomly sampling strategy, and the novelty-seeking exploration bonus as the random network distillation (Burda et al., 2019).

G. More Results

G.1. More Motivating Results

Figure 6 illustrates the area and delay of multipliers vary with the delay constraints from a 3D perspective.

G.2. More Results of Main Evaluation

More details about the minimum area and delay of multipliers can be found at Tables 5 and 6. Through the table, we observe that HAVE achieves the minimum area and delay on each circuit. A noteworthy point is that HAVE and RL-MUL achieve the same performance on the 8-bit booth circuit. We attribute this to the relatively small design space of the 8-bit, making it easier to find the optimal solution. As the design space expands with the increase in scale, the superiority of HAVE which can more efficiently explore and extend the Pareto frontier becomes more pronounced. Especially in 64-bit booth, it can reduce 13.4% on delay and save 9.1% area.

G.3. More Results of Ablation Study

Figure 7 shows the Pareto front of the ablation study. We can find HAVE dominates other methods. And with the increase of the components from RL-MUL to HAVE, the hypervolume continues to grow except for the special case in 8-bit Booth. Table 7 records more details about the hypervolume of 8-booth design tasks compared to Table 1.

	8-bit And		16-bit And		32-bit And		64-bit And	
Method	Hypervolume↑	Improvement(%)↑	Hypervolume↑	Improvement(%)↑	Hypervolume↑	Improvement(%)↑	Hypervolume↑	Improvement(%)↑
Wallace	132	NA	288.53	NA	1805.96	NA	7681.4	NA
GOMIL	130.57	-1.08	301.97	4.66	2229.72	23.46	10312.9	34.26
Random	150.92	14.33	352.01	22.00	2104.02	16.50	8256.41	7.49
RL-MUL	160.84	21.85	433.74	50.33	2543.06	40.81	9663.56	25.80
RL-Scalar	160.84	21.85	434.36	50.54	2528.65	40.02	10136.81	31.97
Have (Ours)	**165.08**	**25.06**	**469.10**	**62.58**	**2970.43**	**64.48**	**11820.00**	**53.88**

Table 3. We calculate the hypervolume of multipliers based on And-Gate. The results demonstrate that HAVE has the largest hypervolume. HAVE produces 16.80% more hypervolume than RL-MUL on average and improves by up to 28.08% compared to RL-MUL, and 64.48% compared to Wallace.

	8-bit Booth		16-bit Booth		32-bit Booth		64-bit Booth	
Method	Hypervolume↑	Improvement(%)↑	Hypervolume↑	Improvement(%)↑	Hypervolume↑	Improvement(%)↑	Hypervolume↑	Improvement(%)↑
Wallace	159.89	NA	220.08	NA	1646.54	NA	6756.84	NA
GOMIL	164.42	2.83	285.34	29.65	1777.03	7.93	9459.48	40.00
Random	184.89	15.64	321.74	46.19	1925.95	16.97	7329.6	8.48
RL-MUL	**187.48**	**17.26**	376.05	70.87	2272.4	38.01	8952.24	32.49
RL-Scalar	**187.48**	**17.26**	369.06	67.69	2263.41	37.46	8642.18	27.90
Have (ours)	**187.48**	**17.26**	**416.07**	**89.05**	**2414.17**	**46.62**	**10544.18**	**56.05**

Table 4. We calculate the hypervolume of multipliers based on Booth-encoding.The results demonstrate that HAVE has the largest hypervolume. HAVE produces 12.59% more hypervolume than RL-MUL on average and improves by up to 23.56% compared to RL-MUL, and 89.05% compared to Wallace.

G.4. More Results of Generalization

Tables 10 and 11 shows the hypervolume of PE arrays. And Tables 8 and 9 records the minimum area and delay found by each method across circuits of different scales.

G.5. More Results of Visualization

In this part, we present visualizations of the compressor trees generated by HAVE and Wallace method. Additionally, Table 12 provides statistics on the number of 3:2 and 2:2 compressors, minimum area, and minimum delay for each compressor tree, facilitating a more comprehensive comparison. In the figure, blue boxes represent compressors common to both CTs, while red rounded boxes represent compressors unique to each CT. Figures 8 to 11 shows the visualization of compressor tree on 8-bit and, 8-bit booth, 16-bit and, 16-bit booth.

8-bit And					16-bit And			
Method	Min Area(μm^2) ↓	Improvement(%)↑	Min Delay(ns) ↓	Improvement(%)↑	Min Area(μm^2) ↓	Improvement(%)↑	Min Delay(ns) ↓	Improvement(%)↑
Wallace	429	NA	0.69	NA	1828	NA	1.26	NA
GOMIL	404	5.83	0.74	-7.25	1705	6.73	1.31	-3.97
Random	404	5.83	0.67	2.90	1787	2.24	1.21	3.97
RL-MUL	**397**	**7.46**	0.65	5.80	1769	3.23	**1.15**	**8.73**
RL-Scalar	**397**	**7.46**	0.65	5.80	1769	3.23	**1.15**	**8.73**
Have (ours)	**397**	**7.46**	**0.64**	**7.25**	**1686**	**7.77**	**1.15**	**8.73**
32-bit And					64-bit And			
Method	Min Area(μm^2) ↓	Improvement(%)↑	Min Delay(ns) ↓	Improvement(%)↑	Min Area(μm^2) ↓	Improvement(%)↑	Min Delay(ns) ↓	Improvement(%)↑
Wallace	7442	NA	2.27	NA	29736	NA	2.95	NA
GOMIL	7034	5.48	2.24	1.32	28567	3.93	2.82	4.41
Random	7401	0.55	2.22	2.20	29728	0.03	2.9	1.69
RL-MUL	7334	1.45	2.13	6.17	29636	0.34	2.78	5.76
RL-Scalar	7371	0.95	2.13	6.17	29702	0.11	2.74	7.12
Have (ours)	**7015**	**5.74**	**2.08**	**8.37**	**28561**	**3.95**	**2.68**	**9.15**

Table 5. We record the minimum area and delay of multipliers based on And-Gate. The results demonstrate that HAVE achieves the minimum area and delay on each circuit design task. Multipliers designed by HAVE achieve a maximum delay reduction of 9.15% and a maximum area reduction of 7.77% compared to Wallace.

8-bit Booth					16-bit Booth			
Method	Min Area(μm^2) ↓	Improvement(%)↑	Min Delay(ns) ↓	Improvement(%)↑	Min Area(μm^2) ↓	Improvement(%)↑	Min Delay(ns) ↓	Improvement(%)↑
Wallace	527	NA	0.96	NA	1867	NA	1.5	NA
GOMIL	517	1.90	0.96	0	1755	6.00	1.49	0.67
Random	519	1.52	**0.9**	**6.25**	1816	2.73	1.41	6.00
RL-MUL	**514**	**2.47**	**0.9**	**6.25**	1768	5.30	1.37	8.67
RL-Scalar	**514**	**2.47**	**0.9**	**6.25**	1782	4.55	1.37	8.67
Have (ours)	**514**	**2.47**	**0.9**	**6.25**	**1719**	**7.93**	**1.35**	**10.00**
32-bit Booth					64-bit Booth			
Method	Min Area(μm^2) ↓	Improvement(%)↑	Min Delay(ns) ↓	Improvement(%)↑	Min Area(μm^2) ↓	Improvement(%)↑	Min Delay(ns) ↓	Improvement(%)↑
Wallace	6539	NA	2.51	NA	24647	NA	3.13	NA
GOMIL	6270	4.11	2.52	-0.40	23782	3.51	2.95	5.75
Random	6535	0.06	2.44	2.79	24626	0.09	3.09	1.28
RL-MUL	6447	1.41	**2.35**	**6.37**	24542	0.43	2.92	6.71
RL-Scalar	6461	1.19	**2.35**	**6.37**	24583	0.26	2.94	6.07
Have (ours)	**6257**	**4.31**	**2.35**	**6.37**	**23773**	**3.55**	**2.85**	**8.95**

Table 6. We record the minimum area and delay of multipliers based on Booth-encoding. The results demonstrate that HAVE achieves the minimum area and delay on each circuit design task. Multipliers designed by HAVE achieve a maximum delay reduction of 10% and a maximum area reduction of 7.93% compared to Wallace.

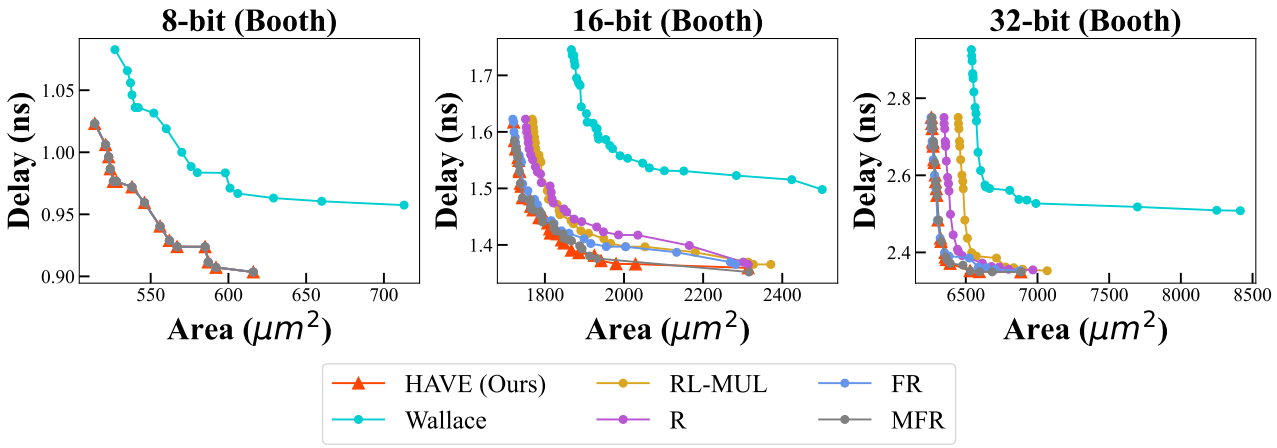


Figure 7. The results demonstrate that the multipliers designed by HAVE Pareto dominate designs by the methods with the increase of components from RL-MUL to HAVE. It shows each component plays an important role in HAVE.

	8-bit Booth		16-bit Booth		32-bit Booth	
Method	Hypervolume↑	Improvement(%)↑	Hypervolume↑	Improvement(%)↑	Hypervolume↑	Improvement(%)↑
Wallace	159.89	NA	220.08	NA	1646.54	NA
RL-MUL	**187.48**	**17.26**	376.05	70.87	2272.40	38.01
R	**187.48**	**17.26**	373.50	69.71	2329.80	41.50
FR	**187.48**	**17.26**	397.32	80.53	2395.31	45.48
MFR	**187.48**	**17.26**	410.34	86.45	2409.01	46.31
HAVE (Ours)	**187.48**	**17.26**	**416.07**	**89.05**	**2414.17**	**46.62**

Table 7. We calculate the hypervolume of the R, FR, and MFR, compared with Wallace, RL-MUL and HAVE. The results demonstrate that each component in HAVE plays an important role in improving the area and delay of designed multipliers.

	8-bit And				16-bit And			
Method	Min Area(μm^2) ↓	Improvement(%)↑	Min Delay(ns) ↓	Improvement(%)↑	Min Area(μm^2) ↓	Improvement(%)↑	Min Delay(ns) ↓	Improvement(%)↑
Wallace	154646	NA	0.69	NA	555527	NA	1.29	NA
GOMIL	148245	4.14	0.74	-7.25	524067	5.66	1.34	-3.88
Random	146406	5.33	0.72	-4.35	545040	1.89	1.25	3.10
RL-MUL	**144500**	**6.56**	0.71	-2.90	540546	2.70	**1.18**	**8.53**
RL-Scalar	**144500**	**6.56**	0.71	-2.90	540546	2.70	**1.18**	**8.53**
Have (ours)	**144500**	**6.56**	**0.68**	**1.45**	**519164**	**6.55**	**1.18**	**8.53**
	32-bit And				64-bit And			
Method	Min Area(μm^2) ↓	Improvement(%)↑	Min Delay(ns) ↓	Improvement(%)↑	Min Area(μm^2) ↓	Improvement(%)↑	Min Delay(ns) ↓	Improvement(%)↑
Wallace	2081967	NA	2.3	NA	7967982	NA	2.97	NA
GOMIL	1977644	5.01	2.27	1.30	7668903	3.75	2.85	4.04
Random	2071617	0.50	2.25	2.17	7965938	0.03	2.93	1.35
RL-MUL	2054320	1.33	2.16	6.09	7942377	0.32	2.82	5.05
RL-Scalar	2063990	0.86	2.16	6.09	7959401	0.11	2.76	7.07
Have (ours)	**1972741**	**5.25**	**2.11**	**8.26**	**7667270**	**3.77**	**2.71**	**8.75**

Table 8. We record the minimum area and delay of PE arrays based on And-Gate. The results demonstrate that HAVE achieves the minimum area and delay on each circuit design task. PE arrays designed by HAVE achieve a maximum delay reduction of 8.75% and a maximum area reduction of 6.56% compared to Wallace.

	8-bit Booth				16-bit Booth			
Method	Min Area(μm^2) ↓	Improvement(%)↑	Min Delay(ns) ↓	Improvement(%)↑	Min Area(μm^2) ↓	Improvement(%)↑	Min Delay(ns) ↓	Improvement(%)↑
Wallace	179773	NA	0.96	NA	566559	NA	1.56	NA
GOMIL	177050	1.51	0.96	0.00	537822	5.07	1.53	1.92
Random	177731	1.14	**0.95**	**1.04**	553484	2.31	1.45	7.05
RL-MUL	**176232**	**1.97**	**0.95**	**1.04**	541363	4.45	**1.4**	**10.26**
RL-Scalar	**176232**	**1.97**	**0.95**	**1.04**	545040	3.80	1.41	9.62
Have (ours)	**176232**	**1.97**	**0.95**	**1.04**	**528834**	**6.66**	**1.4**	**10.26**
	32-bit Booth				64-bit Booth			
Method	Min Area(μm^2) ↓	Improvement(%)↑	Min Delay(ns) ↓	Improvement(%)↑	Min Area(μm^2) ↓	Improvement(%)↑	Min Delay(ns) ↓	Improvement(%)↑
Wallace	1852756	NA	2.55	NA	6667144	NA	3.21	NA
GOMIL	1783843	3.72	2.56	-0.39	6445695	3.32	3.03	5.61
Random	1851666	0.06	2.48	2.75	6661696	0.08	3.15	1.87
RL-MUL	1829195	1.27	**2.39**	**6.27**	6628328	0.58	2.98	7.17
RL-Scalar	1832736	1.08	**2.39**	**6.27**	6650664	0.25	3.02	5.92
Have (ours)	**1781391**	**3.85**	**2.39**	**6.27**	**6443516**	**3.35**	**2.92**	**9.03**

Table 9. We record the minimum area and delay of PE arrays based on Booth-encoding. The results demonstrate that HAVE achieves the minimum area and delay on each circuit design task. PE arrays designed by HAVE achieve a maximum delay reduction of 10.26% and a maximum area reduction of 6.66% compared to Wallace.

	8-bit Booth		16-bit Booth		32-bit Booth		64-bit Booth	
Method	Hypervolume↑	Improvement(%)↑	Hypervolume↑	Improvement(%)↑	Hypervolume↑	Improvement(%)↑	Hypervolume↑	Improvement(%)↑
Wallace	5696.41	NA	57374.84	NA	231219.68	NA	1091876.15	NA
GOMIL	6080.86	6.75	76522.14	33.37	265519.94	14.83	1635968.26	49.83
Random	6393.49	12.24	81141.03	41.42	279267.53	20.78	1234913.46	13.10
RL-MUL	**6612.47**	**16.08**	95165.82	65.87	339493.71	46.83	1578633.59	44.58
RL-Scalar	**6612.47**	**16.08**	94687.18	65.03	337335.67	45.89	1444434.37	32.29
HAVE (Ours)	**6612.47**	**16.08**	**106023.81**	**84.79**	**372530.40**	**61.12**	**1877712.31**	**71.97**

Table 10. We calculate the hypervolume of the PE arrays based on Booth-encoding. The results demonstrate that HAVE has the largest hypervolume. HAVE produces 15.15% more hypervolume than RL-MUL on average and improves by up to 27.39% compared to RL-MUL, and 84.79% compared to Wallace.

	8-bit And		16-bit And		32-bit And		64-bit And	
Method	Hypervolume↑	Improvement(%)↑	Hypervolume↑	Improvement(%)↑	Hypervolume↑	Improvement(%)↑	Hypervolume↑	Improvement(%)↑
Wallace	12511.29	NA	65613.28	NA	344411.77	NA	1493414.77	NA
GOMIL	12010.85	-4.00	68372.12	4.20	450624.39	30.84	2124466.84	42.26
Random	13313.25	6.41	80213.58	22.25	402589.87	16.89	1618902.90	8.40
RL-MUL	15188.23	21.40	101250.37	54.31	476875.33	38.46	2149225.28	43.91
RL-Scalar	15190.40	21.41	101344.04	54.46	473479.45	37.47	2131106.59	42.70
HAVE (Ours)	**16073.69**	**28.47**	**109866.00**	**67.44**	**576194.49**	**67.30**	**2470796.40**	**65.45**

Table 11. We calculate the hypervolume of the PE arrays based on And-Gate. The results demonstrate that HAVE has the largest hypervolume.HAVE produces 17.65% more hypervolume than RL-MUL on average and improves by up to 28.84% compared to RL-MUL, and 67.44% compared to Wallace.

	8-bit And			8-bit Booth			16-bit And		16-bit Booth		
Kinds / Metrics	Wallace	HAVE-1	HAVE-2	Wallace	HAVE-1	HAVE-2	Wallace	HAVE-1	Wallace	HAVE-1	HAVE-2
3:2 Num	37	36	35	29	29	26	197	192	121	119	115
2:2 Num	18	8	14	13	8	17	64	53	64	15	49
Total Num	55	44	49	42	37	43	261	245	185	134	164
Stage Num	4	4	5	3	3	4	6	6	5	4	6
Minimum Delay	0.6938	0.6378	0.6526	0.9652	0.9117	0.9117	1.2626	1.1383	1.4980	1.3593	1.3868
Minimum Area	429	397	407	527	514	521	1828	1769	1867	1723	1792

Table 12. We provide statistics for each compressor tree, including the number of 3:2 and 2:2 compressors, the minimum area, and the minimum delay. The HAVE-1 kind represents the CT with fewer compressors and stages has a smaller area and delay. Conversely, the HAVE-2 kind represents the CT with more compressors and stages but has a smaller area and delay compared with Wallace.

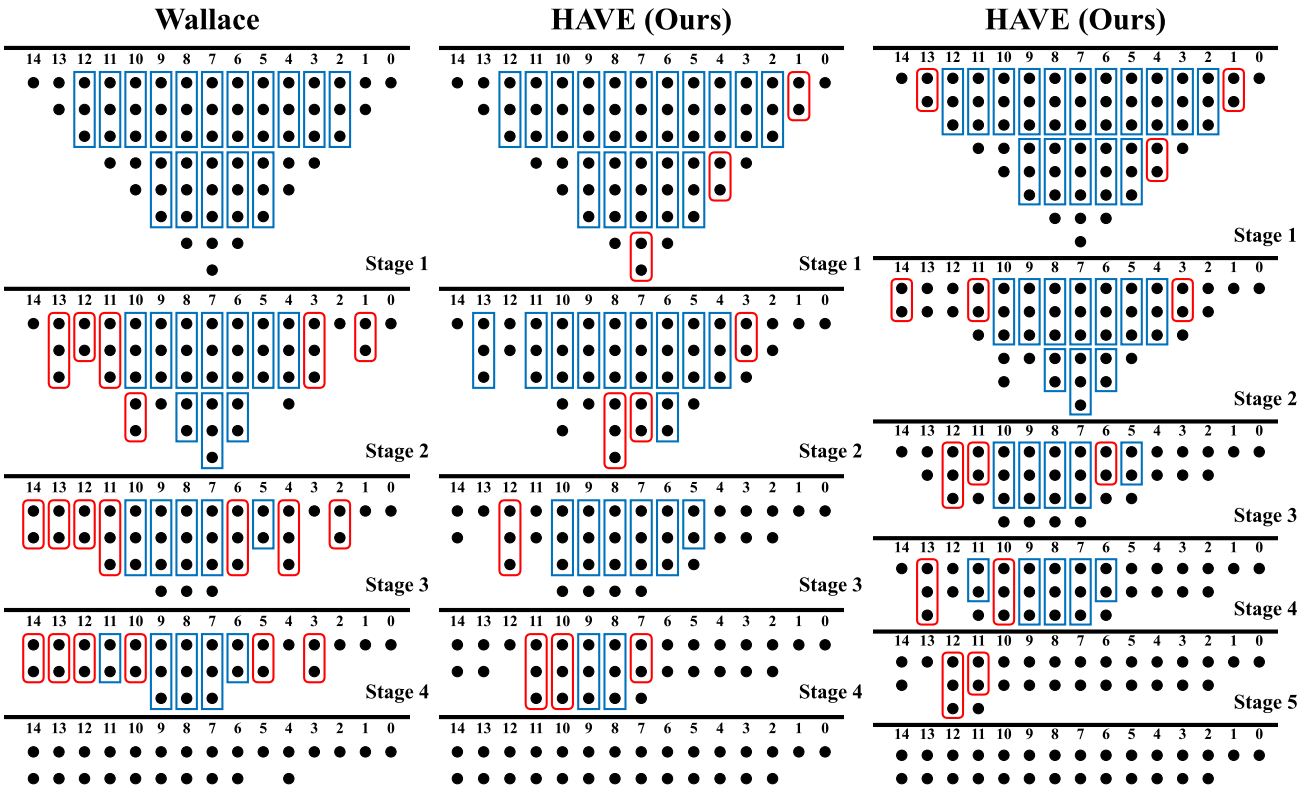


Figure 8. From left to right, they are Wallace, HAVE-1 and HAVE-2 of 8-bit And circuits.

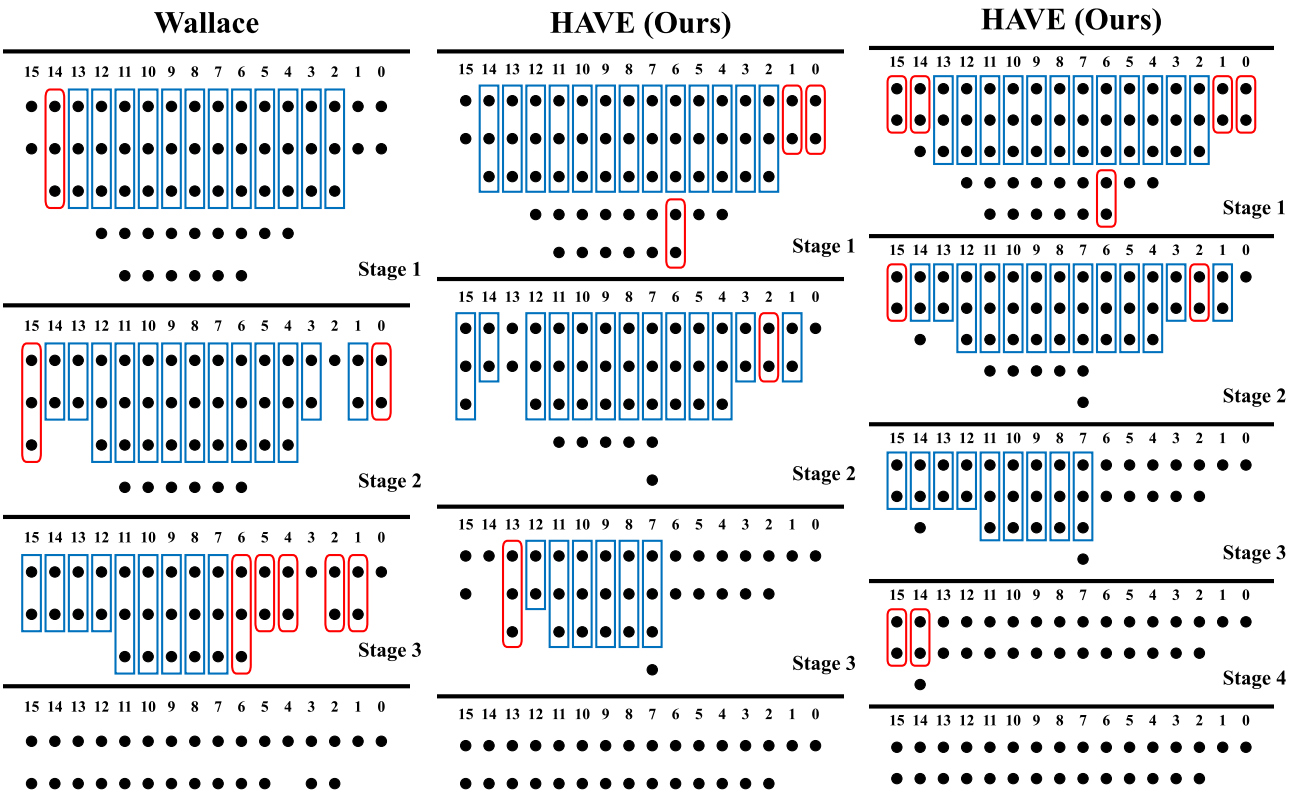


Figure 9. From left to right, they are Wallace, HAVE-1 and HAVE-2 of 8-bit booth circuits.

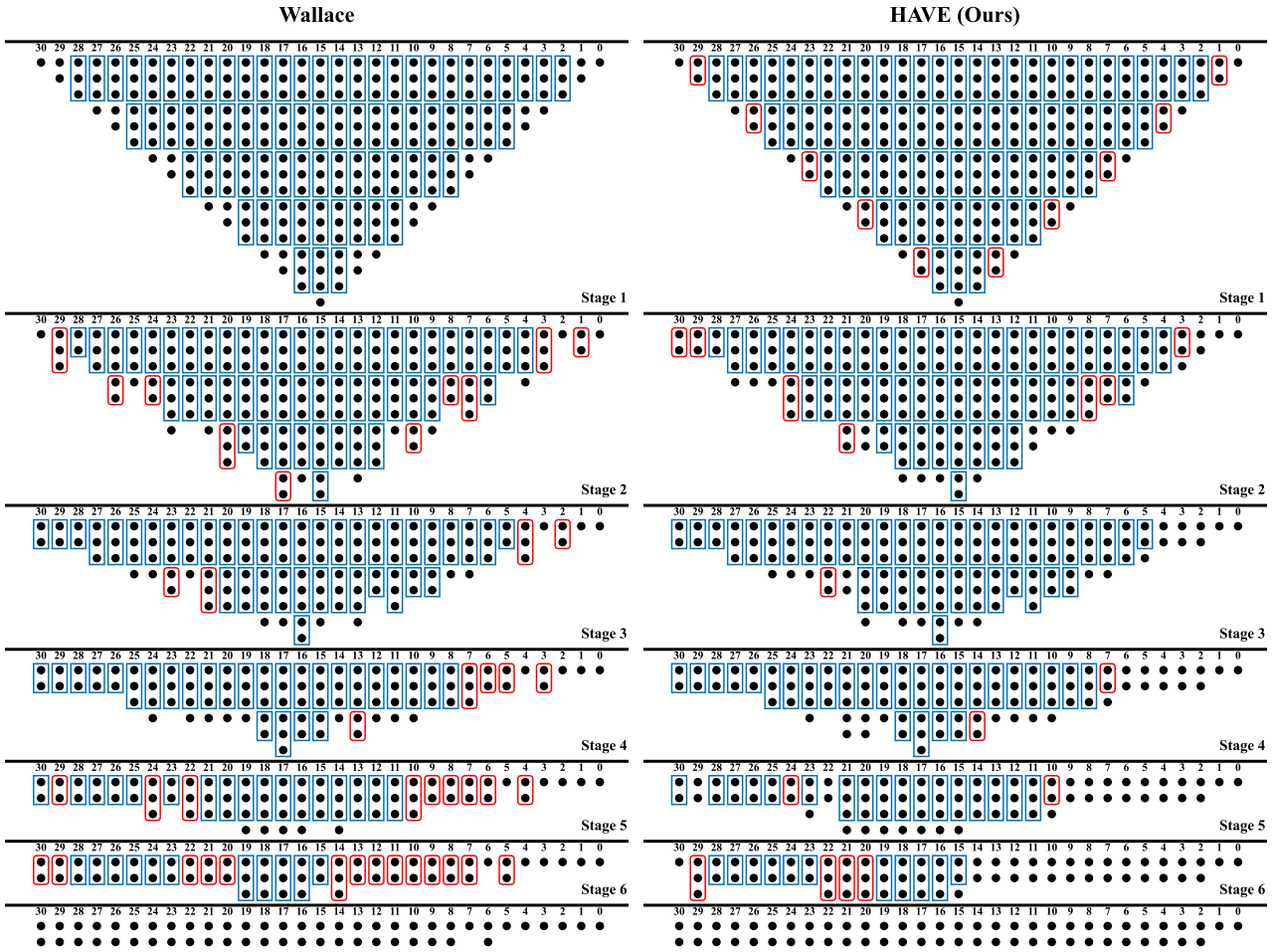


Figure 10. From left to right, they are Wallace, HAVE-1 of 16-bit And circuits.

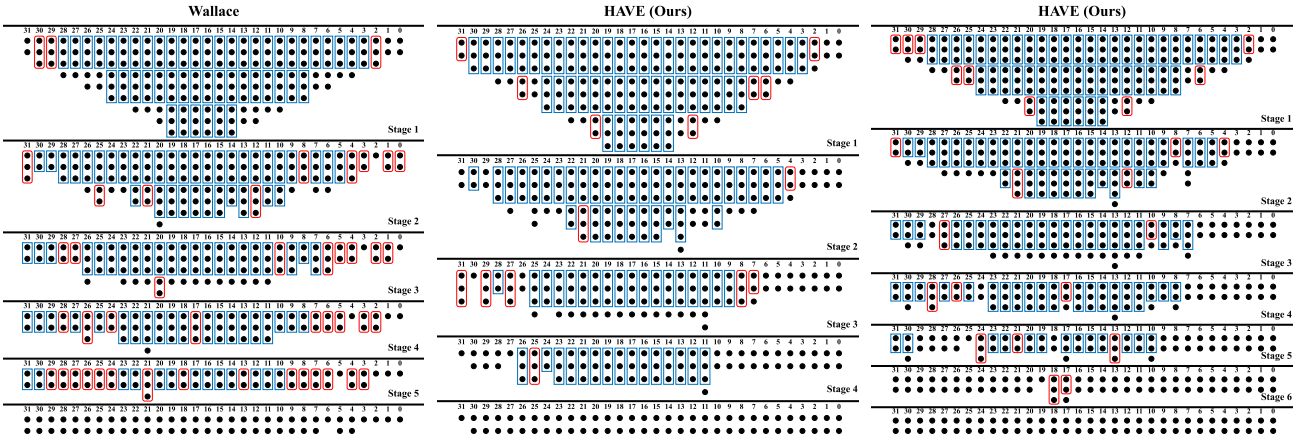


Figure 11. From left to right, they are Wallace, HAVE-1 and HAVE-2 of 16-bit Booth circuits.