RL-Guided Contraction of Symbolic Tensor Networks for Quantum Circuit Equivalence

Suhaib Al-Rousan
Department of Computer Science
Aalborg University
Aalborg, Denmark
suhaibar@cs.aau.dk

Christian Schilling
Department of Computer Science
Aalborg University
Aalborg, Denmark
christianms@cs.aau.dk

Max Tschaikowski
Department of Computer Science
Sapienza University of Rome
Rome, Italy
tschaikowski@di.uniroma1.it

Kim Guldstrand Larsen
Department of Computer Science
Aalborg University
Aalborg, Denmark
kgl@cs.aau.dk

Abstract

Verifying that two quantum circuits are equivalent is a critical challenge in compiler validation. A common approach is to transform the circuits into a tensor network and then iteratively contract tensors in the network. The order in which contractions are applied has a crucial influence on the complexity, as tensor size may grow exponentially in the number of qubits. This motivated research into heuristic policies to find good contraction orders, including recent work based on reinforcement learning (RL). Orthogonally, to mitigate the complexity of an explicit tensor representation, tensor decision diagrams (TDDs) have been proposed as a compact symbolic representation. However, also the TDD performance is sensitive to the contraction order, and policies designed for tensor networks do not perform well in case of a TDD representation. In this paper, we propose an RL framework to obtain a graph-neural-network policy that navigates the trade-off between arithmetic complexity and TDD size. Experimental results demonstrate that a policy learned with our framework is effective at taming the symbolic representation size.

1 Introduction

As quantum hardware scales toward practical utility, ensuring the correctness of the supporting software stack becomes a critical challenge. A central task in this domain is *quantum circuit equivalence checking*: verifying that a compiled or optimized circuit C' is functionally identical to a reference circuit C [24]. Formally, equivalence holds if one circuit is the inverse of the other:

$$C \equiv C' \quad \iff \quad U_C\, U_{C'}^{\dagger} = e^{i\theta} I_{2^n}, \tag{1}$$

where U_C and $U_{C'}$ denote the unitaries implemented by C and C', $e^{i\theta}$ is a global phase (which cannot be observed in practice), and I_{2^n} is the identity matrix on n qubits. Despite the conceptual simplicity of Eq. (1), working with these unitaries is intractable, as they grow exponentially ($2^n \times 2^n$) in the number of qubits n [18].

To mitigate the curse of dimensionality, one can represent quantum circuits as graphs of interconnected tensors, called *tensor networks* (TNs), evaluated via iterative contraction of two tensors [1, 19]. While TNs avoid the exponential size initially, the complexity is shifted to the contraction process: reducing the TN to a final tensor depends critically on the *contraction plan* (the order of contraction). Although

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

all contraction plans yield the same result, intermediate tensors can grow dramatically, making both running time and memory vary significantly. Optimal contraction planning is NP-hard [12].

Tensor decision diagrams (TDDs) combine tensor networks with decision diagrams [5, 11] and provide a powerful alternative to (dense) tensor representations [9]. TDDs use a canonical, graph-based representation with subgraph sharing to compress high-dimensional tensors and can substantially reduce the representation size for structured quantum circuits. However, this representation also changes what makes a "good" contraction plan. Heuristics designed for (dense) TN contraction [8] can perform poorly for TDDs: some contractions may destroy the TDD structure and trigger a node explosion [13]. Consequently, optimizing TDD-based contraction requires explicitly reasoning about the diagram representation and the global topology of the evolving network rather than relying solely on local tensor dimensions.

While reinforcement learning (RL) has recently been applied to (dense) TN contraction [17], that approach relies on static arithmetic proxies (e.g., FLOPs) that can be precomputed from local dimensions. In stark contrast, TDD contraction is a dynamic, symbolic process where a seemingly cheap operation can destroy subgraph sharing and trigger a catastrophic memory explosion.

To address this, we propose a learning-based framework tailored specifically to the TDD regime. We cast the problem as a sequential decision-making process where an agent must navigate the trade-off between computational effort and diagram compression. Unlike static heuristics or prior TN-RL methods, our approach uses a customized *graph neural network* (GNN) to encode symbolic complexities, trained on a dynamic curriculum to stabilize on volatile TDDs. We demonstrate experimentally that our policy produces highly memory-efficient contraction plans. Specifically:

- **Learning approach:** We design a reward function for TDD-based contraction planning that explicitly penalizes peak TDD growth, directly targeting memory bottlenecks.
- **Graph neural network policy:** We develop a GNN policy that extracts structural features from the TDD state, which allows it to generalize to unseen circuits and circuit sizes.
- **Performance:** Our trained agent is competitive with, and even outperforms, state-of-the-art TN-based heuristics on diverse benchmarks, significantly reducing peak memory usage and improving end-to-end contraction time.

Outline. Section 2 reviews the related work. Section 3 provides formal background. Section 4 details our RL framework. Section 5 presents the results. Section 6 concludes the paper.

2 Related Work

We organize the related literature into three areas: (i) decision-diagram representations of quantum circuits, (ii) contraction-planning heuristics, and (iii) RL-based contraction planning.

2.1 Decision Diagrams for Quantum Computing

To mitigate the exponential memory requirements of quantum states and operators, decision diagrams (DDs) are compact, canonical representations as weighted directed acyclic graphs (see Figure 2). Burgholzer and Wille [3] implemented Eq. (1) with DDs to efficiently verify circuit equivalence. Subsequent work enhanced DD-based equivalence checking by incorporating simulation-based counterexamples [4] and by integrating verification with error correction [16]. However, standard DD variants are often less effective when the computation is naturally expressed as a general TN.

TDDs combine the flexibility of TNs with the canonical compression and node sharing of DDs [9]. While TDDs can significantly improve scalability on structured circuits, their practical efficiency, as for ordinary tensors, depends strongly on the order of their contraction.

2.2 Contraction-Planning Heuristics

Selecting a contraction plan for a TN is a central algorithmic challenge and is known to be NP-hard [12]. Typical approaches rely on static heuristics. For standard TNs, state-of-the-art tools such as *Cotengra* (which is the underlying technology in quimb [6]) employ greedy search and graph-based methods to reduce arithmetic cost (e.g., FLOPs) [8].

For TDD-based contraction, the objective shifts to controlling the DD growth, typically measured by *node count*. Heuristics tailored to TDDs have been proposed to address this mismatch. Larsen et al. [13] evaluated an approach that greedily minimizes the TDD size in the next step, which was found too costly; instead, they proposed the *Counting* method, which balances contraction frequency across nodes. *Fast-TDD* and related systems combine optimizations such as rank simplification and caching to accelerate contraction [26, 15]. Yet, static heuristics typically make local choices, not considering how current contractions affect future compressibility, which can lead to excessive blow-ups.

2.3 Reinforcement Learning for Contraction Planning

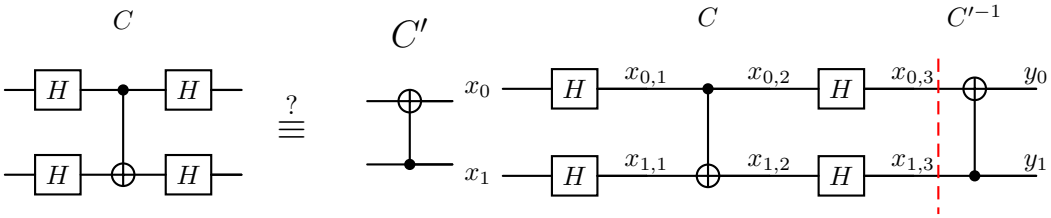
Due to the limitations of hand-crafted heuristics, RL techniques have been applied to contraction planning in (dense) TNs. Our work is inspired by Meirom et al. [17], who trained a GNN policy. However, adapting their approach to TDDs is not straightforward. In TNs, contraction costs are static and can be encoded directly into edge features. For TDDs, the cost (node count) is highly dynamic and depends on which structural symmetries are preserved, thus requiring a different formulation. Our work addresses this gap by designing an RL framework that explicitly targets symbolic memory bottlenecks. Because TDD costs cannot be precomputed, our approach introduces a multi-scale state representation—incorporating node-level memory loads and graph-level global contexts—that allows the agent to predict and avoid memory explosions. Furthermore, we contrast with approaches like Liu and Zhang [14], who proposed an RL approach based on a Hamiltonian formulation for Sycamore-like circuits, by focusing strictly on the unique properties of symbolic decision diagrams.

3 Background

This section briefly introduces quantum circuits, TNs, TDDs, and the contraction-planning problem.

3.1 Quantum Circuits and Equivalence Checking

A quantum circuit C is a finite sequence of quantum gates acting on n qubits. Each gate corresponds to a unitary matrix, and composing gates layer-by-layer induces an overall unitary $U_C \in \mathbb{C}^{2^n \times 2^n}$ [18].



(a) Two equivalent circuits.

(b) Combined circuit represented as a TN with explicit indices. (Note that C' is self-inverse: $(C')^{-1} = C'$).

Figure 1: Visualizing two quantum circuits and their corresponding tensor network representation.

3.2 Tensors, Tensor Networks, and Contraction

Following [9], a tensor $\tau \in \mathbb{C}^{d_1 \times \cdots \times d_k}$ is a multidimensional array of order (rank) k. The dimensions are also called indices. Tensors can have multiple shared indices with each other. A TN is a multigraph where nodes correspond to tensors and edges correspond to shared indices. The basic operation in TNs is *contraction* over shared indices, which merges two connected tensors τ_1 and τ_2. Contraction over the shared index set $\mathcal{Z}$ yields the tensor $(\tau_1 \star_{\mathcal{Z}} \tau_2)(\boldsymbol{x}, \boldsymbol{y}) = \sum_{\boldsymbol{z} \in \{0,1\}^{|\mathcal{Z}|}} \tau_1(\boldsymbol{x}, \boldsymbol{z})\, \tau_2(\boldsymbol{y}, \boldsymbol{z})$, where $\boldsymbol{x}$ and $\boldsymbol{y}$ range over the non-shared indices.

3.3 Quantum Circuits as Tensor Networks

Quantum circuits admit a natural TN representation in which tensors are the gate unitaries and shared indices are the wires [1]. Specifically, a k-qubit gate corresponds to a rank-$2k$ tensor with k

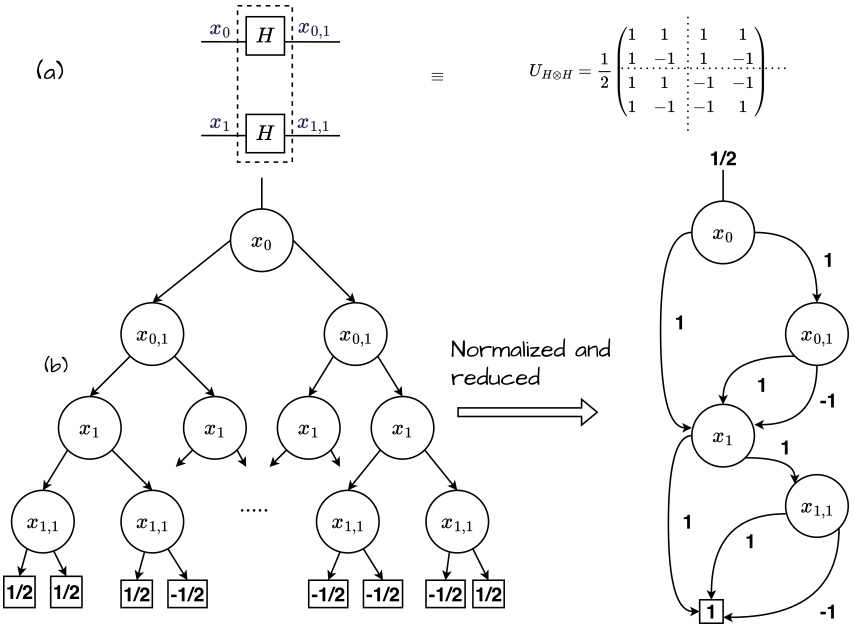


Figure 2: (a) First layer of C (Figure 1(a)) and its unitary. (b) The corresponding TDD. (Reproduced from Zhang et al. [26].)

input/output indices. The unitary composition from Eq. (1) can be applied at the circuit level, yielding a combined circuit $C \circ (C')^{-1}$, which is straightforward to obtain [3].

Example. Figure 1(a) shows a two-qubit circuit C with layers $U_{L_0} = H \otimes H$, $U_{L_1} = \mathrm{CNOT}_{0\to1}$, and $U_{L_2} = H \otimes H$, hence $U_C = (H \otimes H)\mathrm{CNOT}_{0\to1}(H \otimes H)$. This is equivalent to the circuit C' shown in the same figure. Figure 1(b) then illustrates the composed equivalence-checking instance $C \circ (C')^{-1}$. That combined circuit induces a TN (by simply adding the shown wire indices).

3.4 Tensor Decision Diagrams and TDD Networks

TDDs are specialized DDs to compactly represent tensors as rooted, weighted directed acyclic graphs by exploiting shared substructure [9]. In a nutshell, let I be a linearly ordered set of binary indices. A tensor $\tau\colon \{0,1\}^{|I|} \to \mathbb{C}$ admits a Boole-Shannon-style expansion on any $x \in I$:

$$\tau \;=\; (1-x)\,\big(\tau\,|_{x=0}\big) \;+\; x\,\big(\tau\,|_{x=1}\big), \tag{2}$$

where $\tau\,|_{x=b}$ is the slice obtained by fixing $x = b$. Recursively applying Eq. (2) over the global order on I yields a binary TDD whose edges carry complex weights. By merging isomorphic subgraphs and applying normalization rules, TDDs provide a canonical representation (cf. Figure 2).

A *TDD network* (TDDN) is a TN in which each tensor is represented as a TDD. Here, the dominant memory cost is governed by the *number of TDD nodes* created during intermediate contractions, making *peak node count* a natural proxy (cf. Figure 3).

3.5 Contraction Planning and Why It Matters

A *contraction plan* specifies a sequence of pairwise contractions that reduces a TN (or TDDN) to a single tensor (or TDD). While the final result is invariant to the order, intermediate resource usage can differ drastically. For TNs, different plans can change the arithmetic cost by orders of magnitude. For TDDs, contractions may alter the representation size by destroying structural regularities [13].

Example. For tensors $\tau_A \in \mathbb{R}^{1000\times2}$, $\tau_B \in \mathbb{R}^{2\times1000}$, and $\tau_C \in \mathbb{R}^{1000\times2}$, the order $(\tau_A\tau_B)\tau_C$ costs 4×10^6 FLOPs, whereas $\tau_A(\tau_B\tau_C)$ costs only 8×10^3 FLOPs, which is a $500\times$ difference.

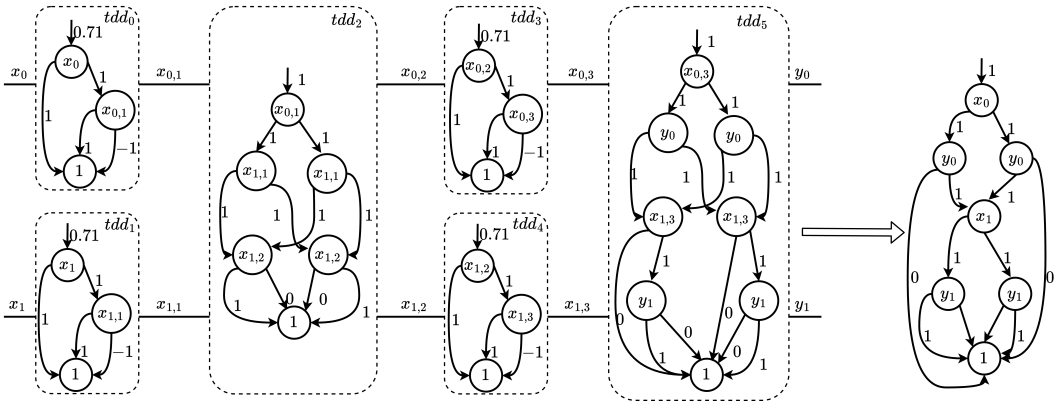


Figure 3: A TDDN representation of the TN in Figure 1(b), and the TDD resulting from contraction.

4 Reinforcement Learning Approach

We cast contraction planning for TDDNs as a *sequential decision-making* problem. Starting from an initial TDDN, an agent repeatedly selects an index to contract, yielding a new (partially contracted) TDDN, until a single TDD remains. This evolving TDDN as the input to our agent, which, due to the graph nature of TDDNs, is a GNN. Next, we formalize the contraction process as a Markov decision process (MDP) whose states are TDDNs. Then, we describe our GNN policy and its optimization.

4.1 MDP Formulation

We model contraction planning as an episodic MDP $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r)$ with the following components.

State space $\mathcal{S}$. A state $s_t \in \mathcal{S}$ corresponds to the current partially contracted TDDN, represented as a graph $G_t = (V_t, E_t)$. Each node $v \in V_t$ represents a TDD, and each edge $(u, v) \in E_t$ indicates that the TDDs corresponding to u and v have shared indices.

Action space $\mathcal{A}$. An action selects two neighboring TDDs to contract. Thus, the feasible action set at step t is $\mathcal{A}(s_t) = \{(u, v) \in E_t\}$ such that action $a_t \in \mathcal{A}(s_t)$ chooses an edge (u, v).

Transition dynamics $\mathcal{P}$. Given $a_t = (u, v)$, let I_u and I_v denote the index sets of the two TDDs and $Z = I_u \cap I_v$ the contracted indices. Contraction replaces u and v by a node w with $I_w = (I_u \cup I_v) \setminus Z$. For every remaining node x, the edge between w and x is determined by the shared indices $I_w \cap I_x$, and its features are recomputed accordingly. The episode terminates when $|V_t| = 1$.

Reward r. We penalize memory spikes using the reward function $r_t = -\lambda \cdot \left(\log_{10}(|\mathrm{TDD}_{t+1}|)\right)^{\beta}$, where $|\mathrm{TDD}_{t+1}|$ is the node count of the TDD produced by action a_t, $\lambda = 10/(|V_0| - 1)$ normalizes by episode length, and $\beta = 1.5$ controls the severity of large intermediate TDDs. The logarithm accommodates TDD sizes spanning orders of magnitude. Thus, our reward function provides stepwise feedback for PPO while remaining sensitive to rare memory spikes.

4.2 State Abstraction and Observation

For enabling the agent to generalize across circuit architectures, we convert the dynamic graph $G_t = (V_t, E_t)$ into a padded observation $o_t = (F_t^n, F_t^e, F_t^g, I_t, |V_t|, |E_t|, \mu_t)$, where $F_t^n \in \mathbb{R}^{N_{\max} \times 4}$, $F_t^e \in \mathbb{R}^{M_{\max} \times 2}$, and $F_t^g \in \mathbb{R}^{1 \times 5}$ are node, edge, and global features, I_t is the adjacency list, and $\mu_t \in \{0, 1\}^{M_{\max}}$ masks invalid contractions. The fixed capacities $(N_{\max}, M_{\max})$ provide a consistent input dimension while still allowing variable-size graphs through padding.

Node features (F_t^n). Each node $v \in V_t$ is encoded by a 4-dimensional feature vector: (i) *Merge count*: The number of original TDDs contracted into v. (ii) *Memory load*: The TDD size ($\log_{10}$ of node count). (iii) *Connectivity density*: The bond dimension (indices shared with all neighbors), normalized by the node buffer capacity $N_{\max}$. (iv) *Node degree*: The number of nodes adjacent to v,

normalized by an empirical scaling factor 10. Together, these features encode the TDD complexity and allow one to distinguish between densely connected and peripheral nodes.

Edge features (F_t^e). Each edge (u, v) is encoded by a 2-dimensional feature vector: (i) *Bond strength*: The number of shared indices between u and v, normalized by 5 to typical bond counts (approximately 1–5 shared indices). (ii) *Local topology*: The average bond dimension of u and v, normalized by the node buffer capacity $N_{\max}$. Together, these features encode the connectivity of u and v relative to the neighbors.

Global features (F_t^g). We use a 5-dimensional global feature vector: (i+ii) Normalized counts of active nodes and edges relative to the maximum capacity. (iii+iv) Average and maximum TDD sizes (log-scaled). (v) A normalized step counter, tracking episode progress.

Example. Consider the TDDN in Figure 3 (left). The graph G_0 contains $|V_0| = 6$ nodes and $|E_0| = 6$ edges, ignoring the open wires x_0, x_1, y_0, and y_1. With capacities $N_{\max} = 87$ and $M_{\max} = 151$, the node *tdd*$_0$ has the feature vector $[1.0, 0.48, 0.01, 0.1]$ (rounded): one original TDD, memory load $\log_{10}(3)$, one shared index normalized by $N_{\max}$, and one neighbor normalized by 10. The edge $x_{0,1}$ has feature vector $[0.2, 0.03]$ (rounded), corresponding to one shared index normalized by 5 and the average local topology normalized by $N_{\max}$. The global vector is $[0.07, 0.04, 0.62, 0.9, 0]$ (rounded): active node/edge ratios, average/maximum log-scaled TDD sizes, and episode progress.

4.3 Neural Network Architecture

We employ an actor-critic architecture that processes an observation vector o_t by a shared graph encoder designed to capture the structural properties of the TDDN. We implement a three-stage hybrid embedding strategy: (i) *Local feature extraction (GINE):* we first apply graph isomorphism network layers with edge features, i.e., GINE-style message passing [10, 25], to extract local geometric and symbolic cues. This allows the agent to distinguish topologically similar neighborhoods that differ in edge-level contraction properties. (ii) *Neighbor prioritization (GAT):* the embeddings are refined using multi-head graph attention layers over the sparse TDDN adjacency, allowing each node to weight its immediate neighbors dynamically [2]. (iii) *Sparse edge-conditioned transformer layer:* we apply a TransformerConv layer over the same sparse edge set, using edge attributes in the attention update. This improves local message passing without introducing dense all-to-all attention.

The resulting shared embeddings feed into two separate heads. The *critic* pools the node embeddings into a graph-level vector, combines it with the processed global features, and estimates $V_\phi(o_t)$ using a multi-layer perceptron (MLP). The *actor* scores each candidate contraction edge $e = (u, v)$ using both learned node embeddings and edge/global context. Specifically, for encoded node embeddings h_u, h_v, encoded edge feature e_{uv}, and processed global feature g_t, it forms $z_{uv} = [h_u \| h_v \| e_{uv} \| g_t]$, maps z_{uv} to a scalar score s_{uv} with an MLP, and obtains the final policy by a masked softmax, $\pi_\theta(a_t = e \mid o_t) = \mathrm{softmax}(s_{uv} + \log \mu_t)$.

4.4 Optimization Objective

We adopt proximal policy optimization (PPO) [23] as the RL backbone because contracting is highly volatile: a single suboptimal action can trigger rapid TDD growth [17], making clipped policy updates useful for stabilizing training. Each training instance is a combined circuit $C \circ (C')^{-1}$ formed from an equivalent circuit pair. Given the policy ratio $\rho_t(\theta) = \pi_\theta(a_t \mid o_t)/\pi_{\theta_{\text{old}}}(a_t \mid o_t)$, we optimize the standard PPO clipped actor objective with a value loss and entropy bonus. Figure 4 outlines the overall architecture. The complete training loop, node-based curriculum, and hyperparameters are detailed in Appendix A and Appendix B.

5 Evaluation

Below, we evaluate our RL agent against state-of-the-art contraction heuristics for TNs and TDDNs.

5.1 Setup

All experiments are conducted on a MacBook Air (2024) with an Apple M3 CPU and 16 GB unified memory; we intentionally use consumer hardware, as TDD contraction is primarily CPU-bound. The

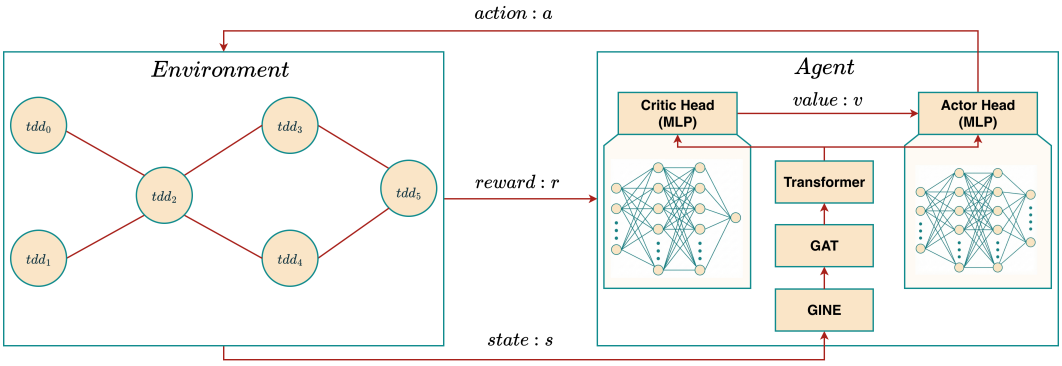


Figure 4: RL framework for TDDN contraction. The environment maintains the TDDN graph, which the agent processes through a graph encoder before actor and critic heads select a contraction edge.

Table 1: Overview of the benchmark families. **Depth** and **Gates** denote the number of layers and gates in the combined circuit $(C \circ (C')^{-1})$ for 5–23 qubits, respectively. **Status** indicates whether the circuit *family* was part of the training curriculum (*partially seen*: the concrete instances were never seen during training) or reserved solely for zero-shot generalization testing (*unseen*).

Family	Depth	Gates	Status
DJ	17–54	49–249	partially seen
GHZ	12–48	12–48	partially seen
W-State	30–138	50–266	partially seen
QNN	28–67	88–395	partially seen
BV	11–29	29–128	unseen

framework is implemented in Python 3.13 using the TDD backend by Hong et al. [9]. To mitigate initial memory bottlenecks, we apply the *Tetris* rank simplification [26] as a preprocessing step, which reduces the initial TN size prior to TDD construction while preserving functional equivalence. The same preprocessing is applied identically to the RL policy and all baselines. We train the agent using PPO with eight parallel environments; all hyperparameters are given in Appendix B.

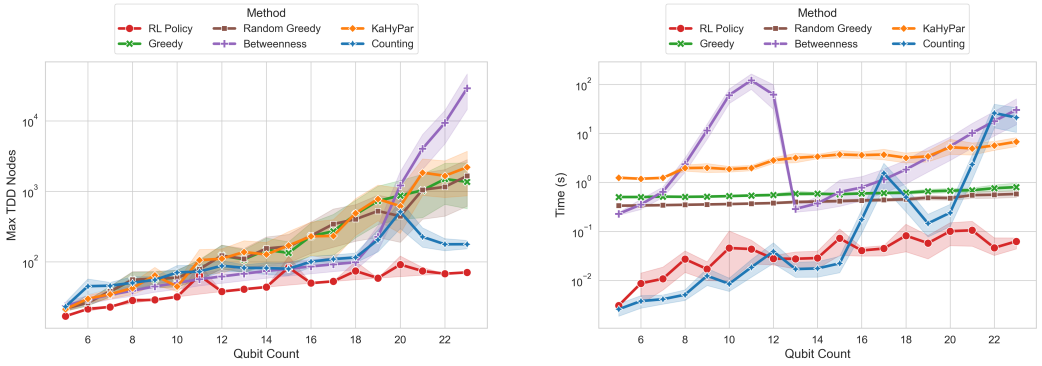
5.2 Benchmarks and Data Sets

We evaluate our approach using five representative circuit families taken from MQT Bench [20]: Deutsch–Jozsa (DJ), GHZ State (GHZ), W-State (W-State), Quantum Neural Network (QNN), and Bernstein–Vazirani (BV). Following the literature, we construct the combined circuit $C \circ (C')^{-1}$ with C the high-level algorithmic circuit and C' a target-dependent implementation at compilation level 3. For each of these circuits, we construct 19 instances with 5 to 23 qubits. Table 1 summarizes the structural properties of the selected families, which exhibit diverse topological features: DJ and BV are dominated by single-qubit gates and sparse connectivity while GHZ, W-State, and QNN feature dense or structured entanglement that significantly increases contraction complexity.

We do not use the BV family during training to assess the agent's generalization to unseen topologies. The training set comprises a random selection of 70% from the 19×4 instances corresponding to the other four circuit families, with the remaining 30% held out for validation. To evaluate whether training exclusively on equivalent pairs biases the planner toward identity-producing contractions, we additionally construct 304 non-equivalent test instances; these are used only for evaluation and are described in Appendix C.5.

5.3 Curriculum Learning

We found it critical to employ a three-phase curriculum to not overwhelm the agent with hard instances early in the training, since choosing a bad contraction plan may lead to exploding memory and running time. For that, we order the training TDDNs by graph size. The first curriculum phase covers the smallest third; its episodes are augmented with TDDNs of small random circuits matched



(a) Peak TDD size.

(b) Total (planning + contraction) running time.

Figure 5: Aggregated performance comparison (averaged) on the evaluation set.

in graph size to this third. The second phase also includes medium-size graphs, with half of its episodes drawn from larger random circuits. The third phase includes the full dataset, with 29.7% of its episodes still drawn from random circuits (Appendix B.4).

5.4 Baselines

We compare our RL-guided contraction strategy against five baselines. Four are TN optimizers from Cotengra [8, 7]: (i) *Greedy*, which contracts the locally cheapest pair of tensors at each step and is fast but myopic, (ii) *Random Greedy*, a stochastic extension that samples among low-cost options to escape local minima, (iii) *Betweenness*, which orders contractions according to edge betweenness centrality and delays merging graph clusters until late in the sequence, and (iv) *KaHyPar*, which uses multilevel hypergraph partitioning [22] for globally informed reasoning at the cost of higher planning time. As a TDD-specific comparison, we include (v) the *Counting* heuristic [13], a lightweight FIFO-style method that contracts edges in order of arrival and does not account for the TDD structure.

While the Cotengra baselines operate on the initial TN and do not use a TDD representation, we still execute their contraction plan using the TDD representation for a fairer comparison. The running time would increase significantly if applied to a dense tensor representation instead.

5.5 Results

We evaluate each method on held-out or otherwise unseen circuits across all benchmark families. Our primary metrics are peak TDD node count and total running time (contraction planning plus TDDN contraction).[1] Figure 5 shows the aggregated results over the filtered evaluation set. For a per-family breakdown, we refer to Appendix C.1, specifically Figure 6 and Figure 7.

Figure 5(a) shows the aggregated peak TDD size. Overall, the RL policy provides the lowest memory consumption across the evaluated families. It reduces mean peak TDD size by $5.34\times$ relative to Greedy, $4.49\times$ relative to Random Greedy, $10.73\times$ relative to Betweenness, $6.50\times$ relative to KaHyPar and $2.51\times$ to the TDD-specific Counting heuristic. Counting remains highly effective on several structured families, such as `DJ`, `GHZ`, and `BV`, where local symbolic cancellation patterns seem easy to exploit, but it is less robust on the `W-State` and `QNN` circuits, where later contraction stages can produce substantially larger TDDs.

Figure 5(b) shows the aggregated total time. Our policy has low inference overhead, with an average total time of approximately 0.047s on the evaluation set, meaning an end-to-end speedup of $63.7\times$ over KaHyPar and $12.3\times$ over Greedy, all while maintaining low peak TDD size. This indicates that the policy amortizes contraction-order search during training to provide fast inference-time decisions.

Training-run robustness. Across four independent training runs (the final policy plus three re-trainings) on the same 114-circuit held-out set, the mean peak TDD size is 48.5 ± 3.1 nodes (mean $\pm$

[1] `https://github.com/srousan18/rl_tddn_contraction_order`

sample std., $n = 4$), with reductions of $5.99 \times \pm 0.39$ over Greedy and $2.39 \times \pm 0.15$ over Counting and a $69.4 \times \pm 0.4$ speedup over KaHyPar. Most variation stems from W-State (Appendix C.7).

Non-equivalent circuit pairs. Since equivalence is unknown at verification time, we also evaluate the final RL policy (trained only on equivalent pairs, no fine-tuning) on 304 proven non-equivalent instances built from compiler-style mutations and `DJ` pairs of different parity. On the 279 instances that every method fully contracts, the RL policy achieves the lowest peak TDD size (mean: 393 nodes vs. 400–509 for the baselines; geometric mean: 66.4 vs. 76.7–81.9), and it fully contracts 296/304 instances (baselines: 286–300). The policy produced no incorrect verdict: all 296 fully contracted TDDs correctly differ from the identity. The remaining eight W-State instances render no verdict: five time out, and in three the final contraction exceeds a 10^5-node memory safeguard (Appendix C.5).

5.6 Ablation Studies

We ablate both the feature representation and the reward exponent β. The full model achieves the lowest average peak TDD size (48.8 nodes) and the fastest average total time (0.0498s). Removing global features causes only mild degradation ($+12.1\%$ peak size), whereas removing the node resp. edge features is much more harmful ($+106.2\%$ resp. $+160.7\%$ peak size). A topology-only variant also underperforms, confirming that graph connectivity alone is insufficient for TDD contraction planning; symbolic node complexity and edge-local information are both necessary. For the reward, $\beta = 1.5$ gives the best trade-off: smaller values under-penalize memory spikes, while larger values over-penalize useful exploratory contractions. Full ablation results are shown in Appendix C.2.

5.7 Scalability and High-Qubit Regimes

The policy uses fixed observation bounds ($N_{\max} = 87$ nodes and $M_{\max} = 151$ edges for the largest training graph) on the graph size (not qubit count). We evaluate scalability in two directions.

Pure zero-shot RL. Although the policy is trained only on circuits with at most 23 qubits, the 45-qubit instances of `DJ`, `QNN`, and `GHZ` (87 nodes, 129 edges) as well as `BV` (65 nodes, 63 edges) remain within these bounds, so we can evaluate how the RL policy transfers to this qubit regime. The policy matches the virtual-best Cotengra baseline's peak TDD size closely with substantially less planning time (Appendix C.3). This transfer is aided by the scale-normalized and log-scaled features and by sparse local message passing, which do not expose the policy directly to the raw qubit count. We therefore view these results as zero-shot generalization in qubit count for structured families whose TDDN graphs remain within the learned observation bounds.

Beyond the observation bounds. W-State graphs grow faster (e.g., 175 nodes and 305 edges at 45 qubits). Here, we investigate a hybrid approach inspired from [17]: we employ the Counting heuristic for initial contractions until the RL policy is applicable (i.e., until $|V| \leq N_{\max}$ and $|E| \leq M_{\max}$), which then performs all remaining contractions. As shown in Table 2, this hybrid approach is able to tame the node explosion much longer than the baselines, and at 50 qubits still completes in 8.2 s with a peak of $15{,}289$ TDD nodes, whereas all tested baselines exceed the 120 s timeout. The corresponding scaling plots are shown in Figure 9 in Appendix C.4.

5.8 Discussion and Limitations

The results suggest that TDD contraction planning is governed more by evolving graph complexity and not merely by qubit count: small dense circuits can be more challenging than larger sparse ones. Our node-based curriculum therefore stabilizes training by exposing the policy to increasingly complex TDDNs. Finally, the improvement over Counting indicates that the policy learns lookahead behavior: it sometimes accepts a locally larger contraction to avoid later symbolic node explosions.

We identify three main limitations. First, the GNN policy uses fixed observation bounds $(N_{\max}, M_{\max})$, so large initial graphs require the hybrid heuristic/RL strategy described above. Second, training incurs an offline cost because the environment must execute TDD contractions to compute the symbolic reward; in our experiments, this required approximately four hours on consumer hardware. Third, the policy is specialized to structured algorithmic circuits relevant to compiler equivalence checking and does not transfer reliably to unstructured random circuits (Appendix C.6).

Table 2: Scalability on W-State circuits using the hybrid approach (120 s end-to-end budget). Cotengra methods are unseedable and run 3 repeats: peak is min–max over completed repeats, time is the median, and $k/3$ marks partial completion. TO = no repeat finished.

Qubits	Method	Peak TDD Nodes	Time (s)	Status
30	Hybrid heuristic/RL (Ours)	**220**	**0.231**	Success
	Random Greedy	16,442–32,949	3.981	3/3
	Greedy	41,103–166,429	13.838	2/3
	KaHyPar	65,346–264,548	62.343	2/3
	Counting / Betweenness	–	–	TO
38	Hybrid heuristic/RL (Ours)	**127**	**0.251**	Success
	KaHyPar	136,726	44.178	3/3
	Greedy / Random Greedy	166,084 / 83,169	26.5 / 24.6	1/3 each
	Counting / Betweenness	–	–	TO
45	Hybrid heuristic/RL (Ours)	**152**	**0.299**	Success
	Greedy	492,233–522,322	85.671	2/3
	Random Greedy	2,185,637	99.789	1/3
	KaHyPar / Counting / Betweenness	–	–	TO
50	Hybrid heuristic/RL (Ours)	**15,289**	**8.448**	Success
	All baselines	–	–	TO

6 Conclusion

We presented a specialized RL framework for contraction planning with tensor decision diagrams (TDDs). Formulated as a sequential decision-making process over dynamic TDD networks, our proposed method targets the two practical bottlenecks of equivalence checking: peak intermediate memory and planning latency. Our feature encoding, sparse graph policy, and node-based curriculum enable the agent to learn contraction decisions that generalize across circuit sizes and families.

Across five quantum benchmark families (DJ, GHZ, W-State, QNN, BV), the learned policy provides the best aggregate trade-off between peak TDD size and wall-clock time on held-out or otherwise unseen evaluation circuits. It reduces mean peak TDD size by $2.51\times$ over the TDD-specific Counting heuristic and by $5.34\times$ over Greedy, while achieving a mean end-to-end speedup of $63.7\times$ over KaHyPar. In particular, the learned policy is more robust on harder families where local contraction rules can lead to large intermediate TDDs.

These results show that learning-based, TDD-aware contraction planners can complement and, sometimes, improve upon hand-crafted heuristics for quantum circuit equivalence checking. Future work will include the integration into compilation pipelines, extending the curriculum to hardware-topology-aware settings, and improving support for highly entangled or unstructured circuit families.

Acknowledgments and Disclosure of Funding

This work was supported, in part, by the Danish e-infrastructure Consortium (DeiC) under reference number 4316-00006B, the Danish National Research Foundation (DNRF) through the Center CLASSIQUE under grant number 187, the Innovation Fund Denmark under reference number 5366-00005B, the Villum Investigator Grant S4OS under reference number 37819, and the grant QUMOBA of Sapienza University under reference number RM125199BAE72522. The authors declare no competing interests.

References

[1] Jacob Biamonte and Ville Bergholm. Tensor networks in a nutshell. *CoRR*, 2017. URL https://arxiv.org/abs/1708.00006.

[2] Shaked Brody, Uri Alon, and Eran Yahav. How attentive are graph attention networks? In *ICLR*. OpenReview.net, 2022. URL https://openreview.net/forum?id=F72ximsx7C1.

[3] Lukas Burgholzer and Robert Wille. Improved DD-based equivalence checking of quantum circuits. In *ASP-DAC*, pages 127–132. IEEE, 2020. URL https://doi.org/10.1109/ASP-DAC47756.2020.9045153.

[4] Lukas Burgholzer and Robert Wille. Advanced equivalence checking for quantum circuits. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 40(9):1810–1824, 2021. URL https://doi.org/10.1109/TCAD.2020.3032630.

[5] Lukas Burgholzer, Antonio Jiménez-Pastor, Kim Guldstrand Larsen, Mirco Tribastone, Max Tschaikowski, and Robert Wille. Forward and backward constrained bisimulations for quantum circuits using decision diagrams. *ACM Trans. Quantum Comput.*, 6(2):13:1–13:21, 2025. URL https://doi.org/10.1145/3712711.

[6] Johnnie Gray. quimb: A python package for quantum information and many-body calculations. *J. Open Source Softw.*, 3(29):819, 2018. URL https://doi.org/10.21105/joss.00819.

[7] Johnnie Gray. Cotengra documentation. https://cotengra.readthedocs.io/en/latest/, 2026. Accessed: 2026-11-04.

[8] Johnnie Gray and Stefanos Kourtis. Hyper-optimized tensor network contraction. *Quantum*, 5:410, 2021. URL https://doi.org/10.22331/q-2021-03-15-410.

[9] Xin Hong, Xiangzhen Zhou, Sanjiang Li, Yuan Feng, and Mingsheng Ying. A tensor network based decision diagram for representation of quantum circuits. *ACM Trans. Design Autom. Electr. Syst.*, 27(6):60:1–60:30, 2022. URL https://doi.org/10.1145/3514355.

[10] Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay S. Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. In *ICLR*. OpenReview.net, 2020. URL https://openreview.net/forum?id=HJlWWJSFDH.

[11] Antonio Jiménez-Pastor, Kim G. Larsen, Mirco Tribastone, and Max Tschaikowski. Forward and backward constrained bisimulations for quantum circuits. In *TACAS*, volume 14571 of *LNCS*, pages 343–362. Springer, 2024. URL https://doi.org/10.1007/978-3-031-57249-4_17.

[12] Chi-Chung Lam, P. Sadayappan, and Rephael Wenger. On optimizing a class of multi-dimensional loops with reductions for parallel execution. *Parallel Process. Lett.*, 7(2):157–168, 1997. URL https://doi.org/10.1142/S0129626497000176.

[13] Christian Bøgh Larsen, Simon Brun Olsen, Kim Guldstrand Larsen, and Christian Schilling. Contraction heuristics for tensor decision diagrams. *Entropy*, 26(12), 2024. URL https://www.mdpi.com/1099-4300/26/12/1058.

[14] Xiao-Yang Liu and Zeliang Zhang. Classical simulation of quantum circuits: Parallel environments and benchmark. 2023. URL http://papers.nips.cc/paper_files/paper/2023/hash/d41b70011dd21ec3de5e019302279551-Abstract-Datasets_and_Benchmarks.html.

[15] Vicente Lopez-Oliva, José M. Badía, and Maribel Castillo. Efficient quantum circuit contraction using tensor decision diagrams. *J. Supercomput.*, 81(1):354, 2025. URL https://doi.org/10.1007/s11227-024-06836-w.

[16] Chao Lu, Navnil Choudhury, Utsav Banerjee, Abdullah Ash-Saki, and Kanad Basu. QuBEC: Boosting equivalence checking for quantum circuits with QEC embedding. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 43(7):2037–2042, 2024. URL https://doi.org/10.1109/TCAD.2024.3361402.

[17] Eli A. Meirom, Haggai Maron, Shie Mannor, and Gal Chechik. Optimizing tensor network contraction using reinforcement learning. In *ICML*, volume 162, pages 15278–15292. PMLR, 2022. URL https://proceedings.mlr.press/v162/meirom22a.html.

[18] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*. Cambridge University Press, 2010.

[19] Feng Pan, Hanfeng Gu, Lvlin Kuang, Bing Liu, and Pan Zhang. Efficient quantum circuit simulation by tensor network methods on modern gpus. *ACM Trans. Quantum Comput.*, 5(4): 26:1–26:26, 2024. URL https://doi.org/10.1145/3696465.

[20] Nils Quetschlich, Lukas Burgholzer, and Robert Wille. MQT Bench: Benchmarking software and design automation tools for quantum computing. *Quantum*, 7:1062, 2023. URL https://doi.org/10.22331/q-2023-07-20-1062.

[21] Antonin Raffin, Ashley Hill, Adam Gleave, Anssi Kanervisto, Maximilian Ernestus, and Noah Dormann. Stable-baseline3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, 22(268):1–8, 2021. URL http://jmlr.org/papers/v22/20-1364.html.

[22] Sebastian Schlag, Vitali Henne, Tobias Heuer, Henning Meyerhenke, Peter Sanders, and Christian Schulz. *k*-way hypergraph partitioning via *n*-level recursive bisection. In *ALENEX*, pages 53–67. SIAM, 2016. URL https://doi.org/10.1137/1.9781611974317.5.

[23] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *CoRR*, 2017. URL http://arxiv.org/abs/1707.06347.

[24] Robert Wille and Lukas Burgholzer. Verification of quantum circuits. In Anupam Chattopadhyay, editor, *Handbook of Computer Architecture*. Springer Nature, 2022. URL https://doi.org/10.1007/978-981-15-6401-7_43-1.

[25] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? In *ICLR*. OpenReview.net, 2019. URL https://openreview.net/forum?id=ryGs6iA5Km.

[26] Qirui Zhang, Mehdi Saligane, Hun-Seok Kim, David T. Blaauw, Georgios Tzimpragos, and Dennis Sylvester. Quantum circuit simulation with fast tensor decision diagram. In *ISQED*, pages 1–8. IEEE, 2024. URL https://doi.org/10.1109/ISQED60706.2024.10528748.

A Implementation

In this section, we provide the pseudocode for our training loop.

Algorithm 1 details the integration of the PPO algorithm with our environment.

A key component is the *node-based curriculum* (Lines 2 and 5). Unlike standard PPO implementations that sample uniformly from the dataset, our approach partitions the benchmark dataset $\mathcal{D}$ into three phases $\{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3\}$ sorted by graph node count. Then we dynamically adjust the difficulty of the sampled circuits based on the training progress t. This ensures the GNN policy π_θ stabilizes its feature extraction on smaller graphs before attempting to solve larger instances. More details are found in Section 5.3.

Algorithm 1 Training GNN Policy with Curriculum PPO

Require: Benchmark Dataset $\mathcal{D}$, max nodes $N_{\max}$, max edges $M_{\max}$, total timesteps T
Ensure: Trained policy π_θ

1: Initialize policy π_θ, value network V_ϕ, and rollout buffer $\mathcal{B}$
2: Partition $\mathcal{D}$ into phases $\{\mathcal{P}_1, \mathcal{P}_2, \mathcal{P}_3\}$ sorted by graph node count
3: $t \leftarrow 0$
4: **while** $t < T$ **do**
5: Determine active phase $\mathcal{P}_k$ based on t ▷ Apply Node-Based Curriculum
6: $C \sim \text{Uniform}(\mathcal{P}_k)$ ▷ Sample circuit from current phase
7: $\mathcal{N} \leftarrow \textsc{CircuitToTDDN}(C)$
8: $G \leftarrow \textsc{InitGraph}(\mathcal{N}, N_{\max}, M_{\max})$
9: **while** $|\textsc{Edges}(G)| > 0$ **do**
10: $o \leftarrow \textsc{Observe}(G)$ ▷ Extract GNN features
11: $a \sim \pi_\theta(o)$ ▷ Sample edge to contract
12: $(G, r) \leftarrow \textsc{Step}(G, a)$ ▷ Contract and compute reward
13: $\textsc{Store}(\mathcal{B}, o, a, r)$
14: $t \leftarrow t + 1$
15: **if** $\mathcal{B}$ is full **then**
16: $\textsc{UpdatePPO}(\pi_\theta, V_\phi, \mathcal{B})$ ▷ Compute advantages and update networks
17: $\textsc{Clear}(\mathcal{B})$
18: **end if**
19: **end while**
20: **end while**
21: **return** π_θ

B Hyperparameters and Reproducibility

Compute resources. All reported experiments were run on a consumer-grade MacBook Air (2024) with an Apple M3 processor and 16 GB unified memory. Training used eight parallel CPU environments and required approximately four hours for the reported model. Evaluation runs were CPU-only and used the wall-clock timeouts stated in Section 5.7. Preliminary exploratory runs used the same hardware class and did not require substantially larger compute resources than the final reported experiments. To facilitate reproducibility, we provide the normalization constants, network architecture specifications, and PPO hyperparameters used in our experiments.

B.1 Feature Normalization

The environment normalizes raw topological features to ensure numerical stability for the neural network. The scaling constants used to bound the observation space are:

- **Node buffer ($N_{\max}$):** Fixed at 87 nodes (determined by the maximum graph size in the training set plus padding).
- **Edge buffer ($M_{\max}$):** Fixed at 151 edges.
- **Max degree scaling (B_n):** 10 (used to normalize node degrees).

Table 3: PPO hyperparameters.

Parameter	Value
Learning rate (α)	1×10^{-4}
Batch size	256
Rollout buffer size (`n_steps`)	2048
Optimization epochs (`n_epochs`)	10
Clip range (ϵ)	0.2
Entropy coefficient (c_e)	0.01
Value function coefficient (c_v)	0.5
Discount factor (γ)	0.99
GAE parameter (λ)	0.95
Max gradient norm	0.5

- **Max bond strength scaling (B_e):** 5 (used to normalize the number of shared indices between tensors). A sensitivity check over the values $\{3, 5, 8\}$ showed that values ≥ 5 change performance by at most about 4%, whereas the smaller normalizer 3 degrades performance by 16–32%; thus, the value 5 is used as a scale normalization rather than a tuned optimum.
- **Log-size scale:** Raw TDD node counts c are transformed via $\log_{10}(c)$ before usage.

B.2 GNN Architecture

Our policy π_θ uses a *hybrid graph neural network* architecture designed to capture local connectivity, edge-conditioned contraction information, and graph-level context. The specific configuration is:

- **Hidden dimension (d_{model}):** 256.
- **Final feature dimension:** 256.
- **Encoder layers (sequential):**
  1. **GINEConv:** 3 layers with sum aggregation and ReLU activation; edge attributes are included through the GINE message-passing update.
  2. **GATConv:** 2 layers with 4 attention heads and concatenated head outputs. Each head has output dimension $d_{\text{model}}/4$, so the concatenated output remains d_{model}.
  3. **TransformerConv:** 1 sparse edge-conditioned transformer layer with 4 heads, applied over the observed TDDN edge set (not dense all-to-all attention).
- **Dropout:** 0.05 applied after each graph convolution block.
- **Residual connections:** Enabled for all graph layers.
- **Global context:** Raw global features are processed by a two-layer MLP and concatenated with the pooled graph embedding.
- **Pooling:** Global mean pooling is used to generate the graph-level embedding from node latents.
- **Critic head:** A linear value head maps the final graph embedding to $V_\phi(o_t)$.
- **Actor head:** For each candidate edge $e = (u, v)$, the actor concatenates the encoded node embeddings, encoded edge feature, and processed global feature as $[h_u \| h_v \| e_{uv} \| g_t]$, then maps this vector to a scalar edge score using a two-layer MLP.

B.3 Training Hyperparameters

We trained the agent using the proximal policy optimization (PPO) algorithm as implemented in `Stable-Baselines3` [21]. Table 3 lists the hyperparameters used for the results reported in Section 5.5.

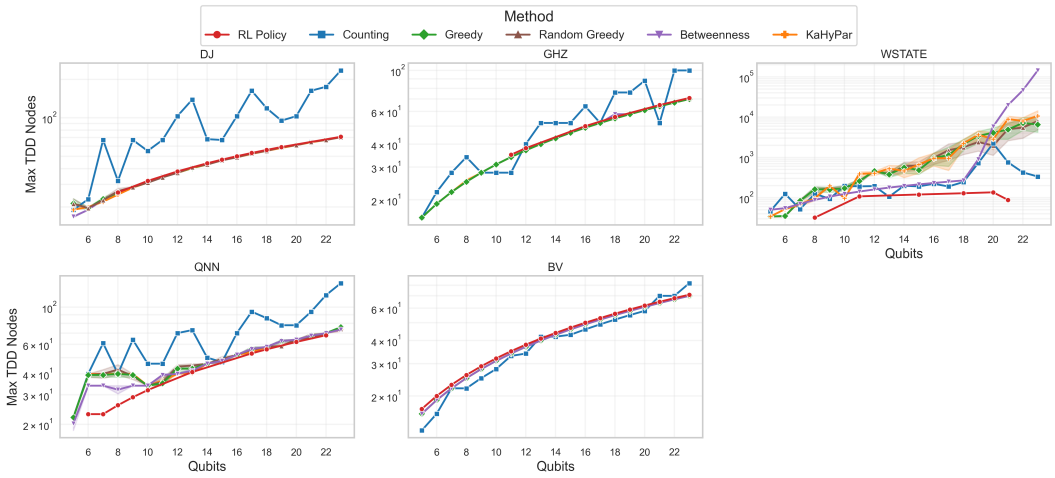


Figure 6: Detailed comparison of peak memory (log scale) across all benchmark families.

B.4 Random curriculum circuits

In addition to benchmark circuits, the curriculum uses randomly generated circuit pairs ("source" and "target") to expose the policy to a wider range of local graph structures. Each random source circuit C is built from the gates that appear in the benchmark source circuits, $\{\texttt{h}, \texttt{x}, \texttt{ry}, \texttt{p}, \texttt{cx}, \texttt{cz}\}$, with rotation angles that are multiples of $\pi/4$, using one of four generic entangling layouts (brickwork, cascade, random-graph, and layered). The target circuit C' is obtained by transpiling C with optimization level 3 to the basis $\{\texttt{rz}, \texttt{sx}, \texttt{x}, \texttt{cx}\}$. The training instance is the combined circuit $C \cdot (C')^{-1}$, which contains the same gates as the benchmark combined circuits, $\{\texttt{h}, \texttt{x}, \texttt{ry}, \texttt{p}, \texttt{cz}, \texttt{cx}, \texttt{rz}, \texttt{sxdg}\}$. The random-circuit generator samples the number of qubits uniformly from 5–23 and a layout depth parameter uniformly from 1–8 in all phases. Of 1,500 generated candidates, those whose TDDN graphs exceed the fixed observation bounds $(N_{\max}, M_{\max})$ are discarded. Instead of using phase-dependent ranges, we select a fixed pool per phase based on graph size. The target sizes correspond to the TDDN graph sizes of the 17 and 34 smallest benchmark training circuits for phases 1 and 2, respectively, and 22 sizes evenly spaced over all 52 benchmark training circuits for Phase 3. For each target, we select the not-yet-selected candidate with the closest graph size whose contraction using the Counting heuristic remains below 2×10^4 TDD nodes and 10 s. This yields 73 random circuits across the three phases (17, 34, and 22, respectively; median combined-circuit depths of 21, 27.5, and 37). The remaining candidates are not used. Following the curriculum, these random circuits constitute 100%, 50%, and 29.7% of the training instances in phases 1, 2, and 3, respectively, with the remainder drawn from the benchmark training circuits. All random circuits are generated only for training augmentation and are not included in the held-out or zero-shot evaluation sets.

C Supplementary Evaluation and Training Dynamics

In this section, we provide additional details about our detailed ablation results and a granular breakdown of the RL policy's performance across individual quantum circuit families.

C.1 Per-Family Performance Scaling

While the main text reports aggregated performance, Figure 6 and Figure 7 provide a per-family breakdown of peak TDD size and total wall-clock time across the evaluated circuit families.

The RL policy shows its strongest relative advantage on the more challenging `W-State` and `QNN` families, where local contraction rules can lead to large intermediate TDDs. On more structured families such as `DJ`, `GHZ`, and `BV`, the *Counting* heuristic is often highly competitive, confirming that TDD-specific local heuristics can be very effective when the circuit topology exposes regular cancellation patterns. However, the learned policy provides more robust behavior across circuit families, avoiding the severe memory growth observed for some baselines on the harder instances.

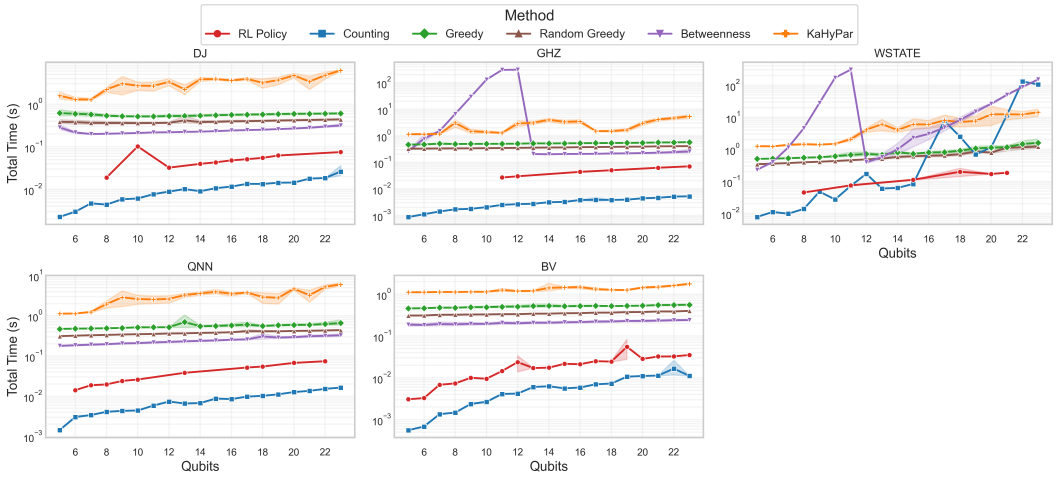
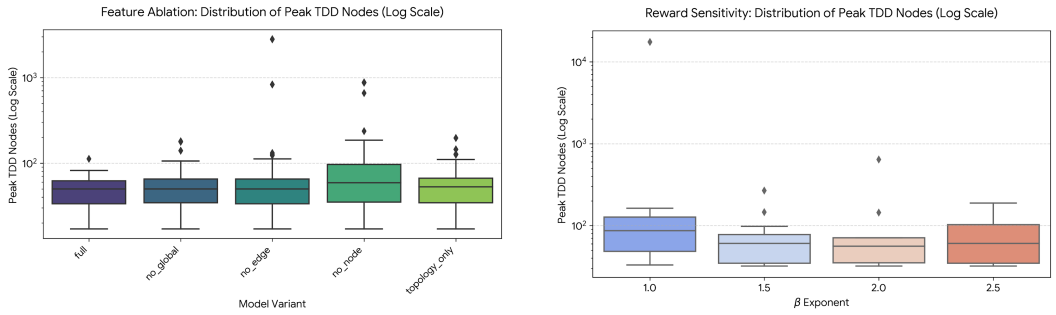


Figure 7: Detailed comparison of total running time (planning & contraction; log scale) across all benchmark families.



(a) Feature ablation: Distribution of peak TDD nodes.

(b) Reward sensitivity over β.

Figure 8: Box plots summarizing the ablation studies (log scale). The distributions confirm the necessity of the full multi-scale feature set to prevent extreme memory blow-ups (left) and validate $\beta = 1.5$ as a good reward exponent among the tested values for stable contraction (right).

The RL policy also avoids the expensive global preprocessing used by partitioning-based baselines such as *KaHyPar*. Since the GINE, GAT, and TransformerConv layers are applied over the sparse observed TDDN edge set only, the per-step neural inference cost scales approximately with the number of observed edges, up to the fixed padding bounds used by the policy.

C.2 Visualizing the Ablation Studies

To visually complement the ablation results presented in Section 5.6, Figure 8 illustrates the impact of our feature and reward designs using logarithmic box plots to capture the performance distribution.

The feature ablation (left plot) highlights the severe degradation in median peak memory and the emergence of extreme outliers (catastrophic memory blow-ups) when critical symbolic features (such as node and edge representations) are removed. This emphasizes that pure graph topology is insufficient for stable TDD planning. The reward sensitivity (right plot) demonstrates the effect of varying the penalty exponent β. It confirms that $\beta = 1.0$ severely under-penalizes memory growth (leading to massive outliers approaching 10^4 nodes), while $\beta = 2.0$ over-penalizes exploration. The exponent $\beta = 1.5$ gives a good balance among the tested values, bounding both the median and worst-case memory peaks.

Table 4: Zero-shot generalization to 45-qubit circuits. The RL policy was trained only on circuits with at most 23 qubits. Peak values are min–max over repeats.

Benchmark	Method	Peak TDD Nodes	Total Time (s)
DJ-45	RL (Ours)	137	0.208
	Counting	475	0.048
	Greedy	**136-155**	0.724
	Random Greedy	**136-179**	0.565
	Betweenness	**136-137**	0.548
	KaHyPar	173	13.288
QNN-45	RL (Ours)	137	0.322
	Counting	265	0.049
	Greedy	**136-184**	0.741
	Random Greedy	**136-172**	0.577
	Betweenness	**136-166**	0.554
	KaHyPar	**136-184**	19.302
GHZ-45	RL (Ours)	137	0.167
	Counting	196	0.010
	Greedy	**136-178**	0.700
	Random Greedy	**136-184**	0.516
	Betweenness	**136**	0.264
	KaHyPar	**136**	11.839
BV-45	RL (Ours)	137	0.083
	Counting	162	0.041
	Greedy	**136-563**	0.829
	Random Greedy	**136-454**	0.663
	Betweenness	**136**	0.281
	KaHyPar	**136**	11.499

C.3 Zero-Shot and High-Qubit Scalability Tables

Table 4 provides additional results for the two scalability regimes discussed in Section 5.7. As a reminder, the 45-qubit DJ, QNN, GHZ, and BV experiments use the final RL policy directly, since their graphs remain within the observation bounds $(N_{\max}, M_{\max}) = (87, 151)$.

C.4 High-Qubit Scalability (W-State)

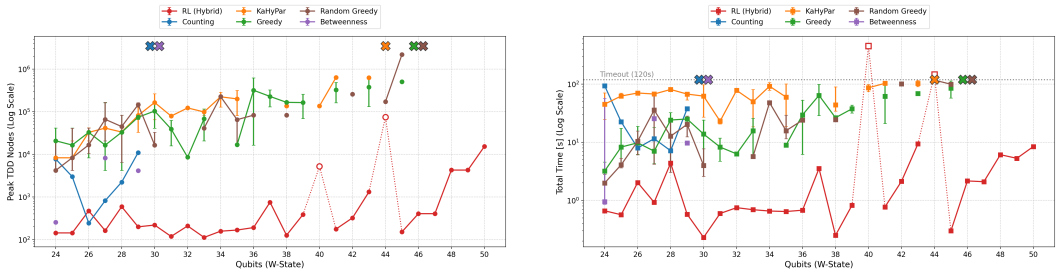
To complement the discussion in Section 5.7, Figure 9 visualizes the performance of our hybrid RL approach on W-State circuits with up to 50 qubits with a 120-second timeout. The exponential divergence in both peak memory and runtime for the baselines clearly demonstrates the necessity of the RL agent's context-aware strategy in higher-qubit regimes.

C.5 Evaluation on Non-Equivalent Circuit Pairs

The policy used throughout the main evaluation is trained exclusively on equivalent circuit pairs. In practical equivalence checking, however, whether two circuits are equivalent is not known in advance. We therefore evaluate whether the learned contraction strategy remains effective for non-equivalent circuit pairs that do not reduce to the identity. All experiments in this subsection use the same final RL policy as the main evaluation; no retraining or fine-tuning is performed.

Dataset construction. We construct 304 non-equivalent instances using two complementary procedures.

Mutation-based instances. We generate 285 instances spanning all five benchmark families and circuit sizes from 5 to 23 qubits. Starting from an otherwise equivalent circuit pair, we inject exactly one error into the composed circuit. The perturbation is one of: (i) deletion of a gate, (ii) reversal of the control and target qubits of a CNOT gate, or (iii) insertion of an X or Z gate. These perturbations are intended to model localized compiler-style errors while preserving the overall structural characteristics of the original benchmark.



(a) Peak TDD Nodes (Log Scale). (b) Total Runtime [s] (Log Scale).

Figure 9: Scalability limits on W-State circuits (24–50 qubits) under a 120 s end-to-end budget per instance (network construction, planning, and the full contraction path). $\times$ markers indicate the first size after which a method never completes within the budget; gaps in a line are sizes where all repeats of Cotengra's unseedable planner timed out but a larger size still succeeded. The hybrid Counting-prefix+RL policy completes 25 of 27 sizes within budget and is the only method that completes beyond 45 qubits; at 40 and 44 qubits (hollow markers) it exceeded the budget and is shown re-measured under a 900 s limit. Peak memory stays 1–4 orders of magnitude below the classical planners throughout, and every completed run was verified to contract to the identity.

Table 5: Performance on equivalent and non-equivalent circuit pairs. Peak values are arithmetic (Avg.) and geometric (Geo.) mean peak TDD node counts over instances solved by every method (112 equivalent, 279 non-equivalent); lower is better. The RL policy is trained only on equivalent pairs and is evaluated without retraining or fine-tuning. *Fully contracted* counts refer to the 304 non-equivalent instances.

	Equivalent Peak		Non-Equivalent Peak		Fully
Method	Avg.	Geo.	Avg.	Geo.	contracted
RL (Ours)	**50.2**	**44.3**	**393.4**	**66.4**	296/304
Counting	113.6	71.2	400.4	81.7	286/304
Betweenness	293.9	53.7	509.4	76.7	293/304
KaHyPar	269.9	58.5	497.1	81.9	298/304
Greedy	237.8	58.6	458.0	80.9	300/304
Random Greedy	182.9	58.3	471.6	81.4	300/304

Cross-instance pairs. We additionally construct 19 Deutsch–Jozsa instances by pairing two genuinely different circuit instances. The two oracle secrets are chosen with different parity so that the circuits implement different functions.

Aggregate contraction performance. Table 5 compares the peak TDD node count on the original equivalent evaluation instances and on the non-equivalent set. All averages are computed on the common instances that every method fully contracts (112 of the 114 equivalent and 279 of the 304 non-equivalent instances), so that no method's average benefits from omitting instances it failed on. Because peaks span more than three orders of magnitude (13 to 65,536 nodes), the arithmetic mean is dominated by a few large W-State instances; we therefore also report the geometric mean. The final RL policy obtains the lowest peak in both settings under both statistics. On the non-equivalent instances, its mean peak is 393.4 nodes, compared to 400.4 for Counting, 458.0 for Greedy, 471.6 for Random Greedy, 497.1 for KaHyPar, and 509.4 for Betweenness; its geometric-mean peak is 66.4 nodes, compared with 76.7–81.9 for the baselines.

The policy's peak increases more than any baseline's when moving from equivalent to non-equivalent pairs: by $1.50\times$ in geometric mean (baselines: $1.15\times$ to $1.43\times$) and by $7.8\times$ in arithmetic mean (baselines: $1.7\times$ to $3.5\times$), the latter driven by a few large W-State instances. Thus, training only on equivalent pairs introduces some sensitivity to non-equivalence, but the degradation is not sufficient to remove the policy's aggregate peak-memory advantage. The effect also depends on the perturbation type: Counting attains a 14% lower geometric-mean peak than the RL policy on the mutations with reversed CNOT gates, while on the 19 cross-instance Deutsch–Jozsa instances, the RL policy is

Table 6: Outcome accounting for the final RL policy on the 304 non-equivalent evaluation instances.

Outcome	Count
Total non-equivalent instances	304
Fully contracted networks / verdicts	296
Correct non-equivalence verdicts	296
False equivalence verdicts	0
Timed out (no verdict)	5
Stopped by 10^5-node safeguard (no verdict)	3

comparable to the Cotengra planners, with relative peak ratios in the range 0.98–$1.02\times$. Hence, the learned policy is not uniformly superior on every non-equivalent instance, but it retains the best aggregate peak-memory behavior over the complete test set.

Completion and verdict correctness. The final RL policy fully contracts 296 of the 304 non-equivalent instances (baselines: 286–300). Every one of these 296 resulting TDDs differs from the identity, in agreement with the independently established ground truth. The remaining eight, all W-State instances, render no verdict: five runs time out; for the other three, the episode ends right before contracting the last two nodes in the network, because that final contraction would produce a TDD of more than 10^5 nodes, which our environment rejects as a memory safeguard. Table 6 summarizes this accounting.

The five timeouts occur on W-State instances: 20- and 22-qubit instances under gate deletion, and 16-, 21-, and 23-qubit instances under CNOT control–target reversal. The three episodes stopped by the node safeguard are W-State instances, with 23 qubits under gate deletion as well as with 17 and 20 qubits under CNOT reversal.

As an additional correctness control, the policy is evaluated on the 114 original equivalent instances used in this comparison. All 114 fully contract and return the identity. Thus, across all instances for which the RL policy renders a verdict, we observe neither a false equivalence verdict on the non-equivalent instances nor a false non-equivalence verdict on the equivalent instances.

C.6 Generalization to Unstructured Random Circuits

The main evaluation considers structured algorithmic circuits, for which the learned policy exhibits strong transfer across circuit families and qubit counts. To probe the distributional boundary of this behavior, we additionally evaluate the final policy on a separate set of fully unstructured random circuits. This experiment is intended as an out-of-distribution stress test.

Evaluation set. We generate 31 random-circuit instances using Qiskit's `random_circuit` generator and transpile them to the gate basis used by the TDD environment. The instances span 5–22 qubits, and all resulting TDDN graphs fit within the fixed policy observation bounds $(N_{\max}, M_{\max})$. Consequently, the policy is applied directly in all cases (i.e., without invoking the hybrid approach). The same final RL policy used in the main evaluation is used here, with no retraining or fine-tuning.

This evaluation set is independent of the random circuits used for curriculum augmentation in Appendix B.4. The latter are generated only during training to diversify local graph structures, whereas the 31 instances considered here are reserved for this out-of-distribution evaluation.

For each random circuit C, we construct an equivalence-checking instance by pairing the circuit with its inverse. Thus, the final contraction should represent the identity, while the intermediate network retains the irregular structure of the randomly generated circuit. We evaluate the RL policy and the same five baseline planners used in the main experiments.

Results. In contrast to the structured benchmark families, the learned policy does not transfer effectively to this fully unstructured setting. Its geometric-mean peak TDD size is 1781 nodes, the largest among the six evaluated planners. KaHyPar achieves a geometric-mean peak of 412 nodes, which is approximately $4.3\times$ smaller. Moreover, the RL policy is never the single best planner on the subset of instances for which all methods complete.

Table 7: Robustness over four independent training runs: the original policy and three re-trainings. Values are arithmetic mean $\pm$ sample standard deviation ($n = 4$) on the same 114-circuit held-out set. Ratios use mean peak TDD nodes for Greedy and Counting and total wall-clock time for KaHyPar against fixed baseline measurements. Higher is better for ratios; lower is better for peak TDD size.

Metric	Across four training runs
Peak TDD nodes	48.5 ± 3.1
Peak reduction vs. Greedy	$5.99 \times \pm 0.39$
Peak reduction vs. Counting	$2.39 \times \pm 0.15$
Runtime speedup vs. KaHyPar	$69.4 \times \pm 0.4$

These instances are substantially more difficult for all evaluated planners than the structured benchmark circuits: at comparable qubit counts, peak TDD sizes are typically one to two orders of magnitude larger.

One possible contributing factor is the construction of the test pairs. Because each random circuit is paired with its own inverse, the resulting network has a strong mirror structure that conventional planning methods may be able to exploit without having learned the underlying random-circuit distribution. We therefore interpret this experiment primarily as evidence that the current policy should not be expected to generalize without adaptation to arbitrary unstructured circuit families.

Overall, the combined results suggest a clear applicability boundary. The policy transfers effectively across the structured algorithmic circuits studied in the main paper, including previously unseen instances, larger qubit counts, and non-equivalent perturbations, but this transfer does not extend automatically to fully unstructured random circuits.

C.7 Robustness Across Independent Training Runs

The main experiments use a single RL policy checkpoint. To assess whether the reported performance depends strongly on stochastic variation during training, we repeat the complete training pipeline three additional times using seeds 1000–1002. Together with the original RL policy, which was trained without an explicitly fixed seed, this yields four independent training runs. All runs use the same architecture, curriculum, hyperparameters, and evaluation protocol.

Aggregate robustness. We evaluate all four trained policies on the same 114-circuit held-out set used in the main evaluation. Table 7 reports the arithmetic mean and sample standard deviation across the four runs.

The aggregate peak TDD size varies by approximately 6.4% relative to its mean, indicating that the principal performance trend is stable across independent training runs. The relative advantages over Greedy and Counting also remain consistent, as does the runtime advantage over KaHyPar. These results indicate that the improvements reported for the original policy are not specific to a single stochastic training trajectory.

Per-family variability. The aggregate variation is not distributed uniformly across benchmark families. Nearly all training-run sensitivity is concentrated in W-State, for which the mean peak TDD size is 70.4 ± 18.6 nodes, corresponding to approximately 26.5% relative variation. For every other benchmark family, the corresponding variation is at most 0.4%. This is consistent with the behavior observed elsewhere in the evaluation: W-State exhibits substantially more volatile intermediate TDD growth and is therefore more sensitive to small changes in the learned contraction policy.

Overall, the four-run experiment shows that the aggregate gains are robust to stochastic training variation, while also identifying W-State as the main source of result variability.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes].

   Justification: The abstract and introduction state the paper's scope as RL-based TDD contraction planning for quantum circuit equivalence checking and summarize the empirical contributions. The claims are supported by the method, evaluation, ablation, scalability, and limitations sections.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes].

   Justification: Subsection 5.8 discusses fixed observation bounds, the offline training cost, and the restricted scope of the evaluation to structured benchmark circuits. It also identifies challenging highly entangled or unstructured circuits such as `qftentangled`.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A].

Justification: The paper is primarily methodological and empirical, and it does not claim new formal theorems requiring proof.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes].

Justification: The paper describes the benchmark construction, baselines, metrics, model architecture, reward design, training procedure, curriculum, and hyperparameters. Additional reproducibility details are provided in Appendix A and Appendix B.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in

some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes].

   Justification: The submission includes code and data in the supplementary material, as stated in the experimental section. The appendix provides the necessary hyperparameters, architecture details, and normalization constants for reproducing the reported experiments.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes].

   Justification: Section 5 describes the benchmark families, baselines, held-out evaluation, and metrics. Appendix A and Appendix B provide the curriculum, model architecture, feature normalization, PPO optimizer details, and hyperparameters.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes].

   Justification: Figure 5 reports results with 95% confidence intervals, and the text identifies this as the uncertainty measure. The supplementary ablation plots further show distributional behavior using box plots rather than relying only on point estimates.

   Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes].

   Justification: Section 5.1 reports the hardware used for all experiments and states that TDD contraction is primarily CPU-bound. The same section reports the use of eight parallel PPO environments, while Appendix B provides the full hyperparameter and reproducibility details, including the approximate compute budget for training and evaluation.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes].

   Justification: The work studies algorithmic planning for quantum circuit equivalence checking using synthetic and public benchmark circuits. It does not involve human subjects, private data, surveillance, or deployed decision-making systems.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A].

    Justification: The work is foundational research on quantum circuit verification and contraction planning, with no direct deployment on people or society-facing decision systems. Its main expected impact is improved reliability and efficiency of quantum software tooling.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A].

    Justification: The paper does not release a high-risk model, scraped dataset, or generative system with misuse risks of the type described in the checklist. The released artifacts are code, benchmark-generation scripts, and experimental data for quantum circuit contraction planning.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: The paper uses and cites existing open-source tools and benchmark assets, including MQT Bench [20], Cotengra [8, 7], and Stable-Baselines3 [21]. MQT Bench and Stable-Baselines3 are distributed under the MIT License, while Cotengra is distributed under the Apache License 2.0; the licenses permit the research use described in this paper, subject to their attribution and redistribution terms.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes].

Justification: The paper introduces and submits code and experimental artifacts for the proposed RL contraction planner. The supplementary material and appendix document the architecture, hyperparameters, normalization constants, training procedure, and benchmark construction needed to use these assets.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A].

Justification: The paper does not involve crowdsourcing, user studies, human annotation, or research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.

- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A].

    Justification: No human subjects, crowdsourcing, or participant data are involved, so IRB approval or equivalent review is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for editing or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [N/A].

    Justification: The core method does not use LLMs as a scientific, methodological, or experimental component. Any use of LLMs, if applicable, was limited to writing, editing, or formatting assistance and did not affect the methodology, experiments, or originality of the research.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.