

# MinFill: Reinforcement Learning and GNN Guided Reordering for Fill-In Reduction in RF Circuit Matrices

Hao Zhang<sup>1</sup>, Dan Niu<sup>2</sup>, Cheng Zhuo<sup>3</sup>, Zhou Jin<sup>3\*</sup>

<sup>1</sup>Super Scientific Software Laboratory, China University of Petroleum-Beijing, China

<sup>2</sup>School of Automation, Southeast University, China

<sup>3</sup>College of Integrated Circuits, Zhejiang University, China

\*Corresponding author: z.jin@zju.edu.cn

**Abstract**—Transistor-level circuit simulation is critical to radio-frequency (RF) design, with sparse matrix factorization dominating the runtime. The elimination order (reordering) largely determines fill-ins during factorization, thereby directly impacting floating-point operations and overall performance. However, prevailing reordering methods typically treat matrices as homogeneous graphs and fail to exploit the block-aware structure, which is dense within blocks and sparse across blocks, and is common in RF matrices. This results in limited fill-in reduction and high preprocessing and memory overheads for million-order matrices. In addition, learning methods that lack block-level semantics and legality constraints often yield suboptimal or even invalid eliminations at inference. Therefore, in this paper, we propose MinFill, a block-aware GNN-RL enhanced reordering framework. MinFill reconstructs the coefficient matrix as a tri-relational E/D/F (empty/diagonal/full) block graph, uses a multi-relational GraphSAGE encoder to fuse local density and cross-block coupling features, and trains a maskable PPO policy with a “negative incremental fill-in” reward and joint min-degree and min-potential-fill masks to stabilize learning of sparsity-friendly elimination orders. On six industrial RF matrices, MinFill reduces fill-ins by 22–39%, lowers peak memory by 12–29%, shortens factorization time by about one-third on average, and accelerates the reordering stage by 61.9× on average, delivering significant end-to-end speedups without exhaustive search.

**Index Terms**—Matrix reordering, Sparse linear solver, Reinforcement learning, Graph neural networks, Circuit simulation

## I. INTRODUCTION

In industrial RF and microwave circuit simulations (including multitone steady state and harmonic balance analyses), numerical solvers must repeatedly factor million scale sparse linear systems [1]. These systems exhibit a characteristic *dense within blocks, sparse across blocks* pattern [2]. First, a frequency-domain formulation expands each circuit variable into  $(2H + 1)$  Fourier coefficients, where  $H$  denotes the maximum harmonic order, inflating each scalar coupling into a  $(2H + 1) \times (2H + 1)$  submatrix that is often dense due to inter-harmonic interactions. [3], [4]. Second, RF front ends are architected from highly modular functional blocks (amplifiers, mixers, filters), whose intramodule connections are dense whereas intermodule links are scarce, further reinforcing the interblock sparsity. When such a matrix is subjected to direct

LU factorization, a pivot order that ignores block semantics triggers a cascade of fill-ins as soon as the initial dense blocks are eliminated [5], causing the  $L$  and  $U$  factors to balloon in memory footprint and runtime. Consequently, devising an efficient block-aware reordering strategy prior to factorization has become the critical bottleneck for accelerating numerical solvers in RF simulation flows.

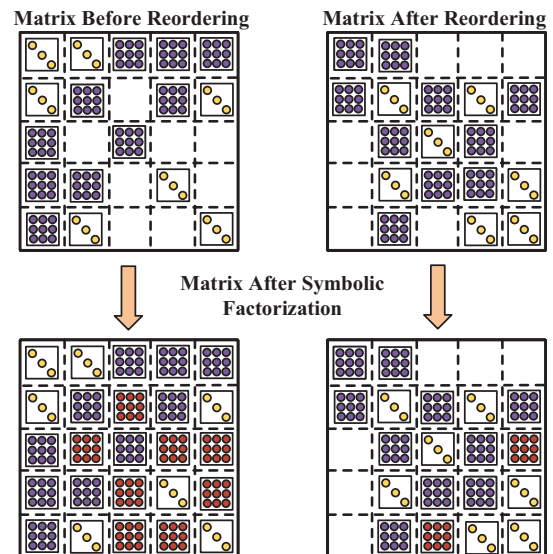


Fig. 1: An improperly eliminated meta-order can result in a large number of fill-ins.

Approximate Minimum Degree (AMD) [6] and Nested Dissection (ND) [7] remain the de facto static strategies for mitigating fill-ins in industrial solvers. These heuristics rely on local, degree- or cut-based decisions and consequently cannot differentiate dense sub-blocks from truly sparse regions. When matrix dimensions exceed one million and the proportion of dense blocks rises, both AMD and ND encounter a dual bottleneck: prolonged preprocessing times and limited fill-in suppression. Recent studies have introduced graph neural networks (GNN) or reinforcement learning (RL) to learn elimination sequences automatically [8]–[14]. However, most

of these models treat the matrix as a homogeneous, untyped graph and overlook the block-level semantics characteristic of RF scenarios, which leads to poor generalisation and illegal actions during inference. A reordering framework that explicitly recognises dense blocks while globally balancing fill-in cost and computational overhead is therefore essential for realising the full performance potential of large-scale RF circuit simulation.

To address these issues, in this paper, we propose a novel sparse matrix reordering method for minimal fill-ins, called MinFill. MinFill embeds the strict block structure inherent in the elimination process of a sparse matrix into a reinforcement learning environment, leverages graph neural networks to extract structural features, and automatically learns an effective reordering strategy through policy optimization. The main contributions of this work are as follows:

- 1) We introduce a block-level matrix modelling scheme that abstracts the original coefficient matrix into a block-node graph and accompanies it with a precise fill-in evaluation rule for block elimination, thereby enabling policy learning on inter-block multi-relational representations while supplying an accurate, step-wise reward signal that reflects the newly introduced block fill-ins.
- 2) We design a multi-relational graph neural network strategy, which combines the two kinds of relations: diagonal, and full blocks embedded in a message passing network to learn a more optimal reordering strategy.
- 3) We use the Maskable PPO algorithm [15] to train an agent to learn the optimal sequence of elimination elements stably and efficiently by masking illegal actions as the action space dynamically decreases with the elimination elements.

Across six RF matrices, MinFill achieves 22.2% and 38.9% average fill-in reductions relative to ND and AMD, respectively; memory drops by 12.2% vs ND and 28.6% vs AMD; factorization is faster by  $1.21\times$  vs ND and  $1.53\times$  vs AMD on average (up to  $2.12\times$ ); and reordering accelerates by  $61.9\times$  vs ND and  $55.4\times$  vs AMD on average (up to  $102.6\times$ ). Results are consistent across the benchmark suite, without per-instance tuning or retuning.

## II. BACKGROUND AND RELATED WORK

### A. Matrix Structure and Solving Challenges in RF Circuit Simulation

RF circuit simulation is an indispensable stage in modern electronic design automation. The nodal admittance (or conductivity) matrix arising from RF analysis often exhibits an inherent block structure due to the circuit’s modular topology and multiport frequency-domain couplings. In such matrices, several locally dense sub-blocks correspond to tightly coupled subcircuits or harmonic components, whereas the overall matrix remains sparse, which is commonly described as “dense within blocks and sparse across blocks.” In practical RF circuits, *fully dense blocks* ( $F$ ) typically originate from nonlinear functional modules, such as low-noise amplifiers,

mixers, and other active stages, whose small-signal Jacobian matrices are close to full. In contrast, *diagonal-band blocks* ( $D$ ) arise mainly from linear passive networks (e.g., filters and matching sections), whose couplings remain near banded after nodal stamping. By explicitly exploiting this structure through chunking algorithms or block-level reordering, matrix factorization can be made more efficient, leading to shorter factorization time and a smaller memory footprint.

To leverage these properties, a range of dedicated sparse linear solvers has been developed for circuit matrices [16]–[20], including NICSLU [21], KLU [22], and SFLU [23], alongside variants tailored specifically to RF block patterns [24], [25]. Techniques such as block triangular form (BTF) extraction and combined symmetric/asymmetric reorderings allow these solvers to exploit structural regularities. However, as circuit sizes and frequency points continue to grow, the elimination process introduces substantial fill-ins, causing memory consumption to escalate rapidly and eroding the performance benefits of existing methods. Consequently, there is an urgent need for smarter reordering and decomposition strategies that further suppress fill-in, alleviate memory pressure, and unlock truly large-scale RF simulation.

### B. Sparse Matrix Numerical Solvers and Elimination Orderings

The fill-in problem generated during Gaussian elimination makes reordering a necessary preprocessing step. For a long time, researchers have proposed a large number of heuristic algorithms (such as Minimum Degree Heuristic (MMD) [26], [27], AMD [6], and Nested Dissection [7], [28]–[30], etc.) to generate a better order of elimination and thus reduce the fill-ins in LU factorization [31]–[34], as shown in Fig. 2. These methods, which reduce fill-ins by representing matrix nonzero patterns as graphs with local cancellation or recursive division of nodes, have been widely used in numerous direct solvers. However, these heuristics often yield suboptimal solutions due to the problem’s NP-hard nature and their dependency on specific matrix structures.

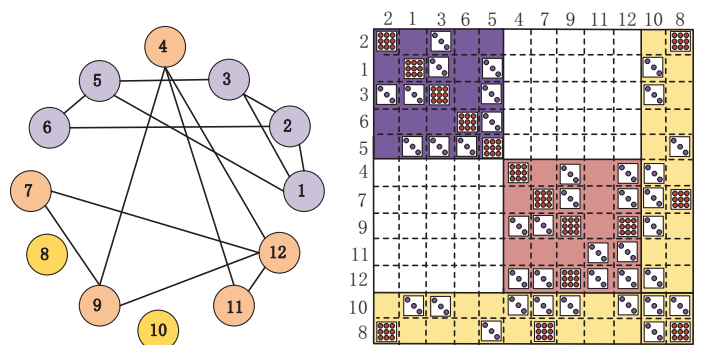


Fig. 2: An example of classical Nested Dissection reordering. (a) The original 12-node graph with three levels of recursively identified separators (purple, peach, yellow) that balance the two sub-domains at each level. (b) Sparse matrix permuted by the ND order.

### C. Sparse Matrix Reordering Methods Combined With Machine Learning

Recent advances in machine learning have opened a new avenue for tackling the sparse matrix reordering problem. Early studies treat reordering as an ordinal decision process in which an agent (or model) incrementally constructs an elimination sequence while minimising the prospective fill-in [14]. For example, the AlphaElimination framework formalises reordering as a single-player game and combines Monte-Carlo tree search (MCTS) [35] with a strategy-value network, achieving a notable fill-in reduction [13]. Gatti et al. employ deep reinforcement learning to solve the graph partition sub-problem in nested dissection, producing high quality vertex separators [12]. Beyond RL, Xia et al. visualise sparse matrices as parallel chunks and feed them to a convolutional neural network to select the most suitable classic reordering algorithm for each matrix, reaching up to 95% of the theoretical optimum while realising an  $11.35\times$  speed-up over exhaustive search [36].

Although these approaches demonstrate the promise of ML in sparse reordering, they typically overlook the strict block structure inherent to RF matrices and often rely on computationally expensive searches (e.g., MCTS), which hinders direct deployment to large-scale industrial problems. Hence, there remains a pressing need for an ML framework that explicitly exploits block structure while learning high-quality reordering strategies with modest computational overhead.

## III. ALGORITHM

This section describes the MinFill method in detail, including four parts: overall flow, block-level problem modelling and fill-in calculation rule, multi-relational GNN policy network and maskable PPO training mechanism.

### A. Overall Framework

In the MinFill framework, we first exploit the block structure of RF sparse matrices by preprocessing them into empty (E), diagonal (D), and full (F) blocks, thereby reducing the state space. The reordering problem is then formulated as sequential elimination on this block graph, with fill-in costs updated by defined rules. To search for an optimal elimination order, we use a multi-relational GNN to extract block features and generate node embeddings from E/D/F edges, producing candidate probabilities via softmax. Finally, a PPO agent with action mask ensures only legal eliminations are considered, while the reward is set as the negative fill-ins, guiding the policy toward sparsity-preserving orders, as shown in Fig. 3.

### B. Block-level Graph Modelling and Fill-in-Aware Reward

Sparse coefficient matrices obtained from RF circuit simulation naturally exhibit a “dense within blocks, sparse across blocks” pattern. To exploit this prior, we partition the matrix, guided by circuit hierarchy or automatic clustering, into a  $p \times p$  block layout with  $p \ll n$ , where  $n$  is the original matrix order.

The  $p$  diagonal sub-blocks  $B_{ii}$  are collapsed into graph nodes

$$V = \{v_1, v_2, \dots, v_p\},$$

whereas every off-diagonal sub-block  $B_{ij}$  ( $i \neq j$ ) becomes an edge  $(v_i, v_j)$ . Edges are labelled by the sub-block pattern:

$$\begin{aligned} E \text{ (empty)} &: \text{no edge,} \\ D \text{ (diagonal)} &: D\text{-edge,} \\ F \text{ (full)} &: F\text{-edge.} \end{aligned}$$

The resulting multi-relational block graph is much smaller than the element-level graph yet keeps the density semantics essential for reordering.

When a node  $a$  is eliminated, new edges may appear among its neighbours  $N(a)$ . We denote the neighbourhood size and its current edge count by

$$d_a = |N(a)|, \quad e_a = |E(N(a))|. \quad (1)$$

At most

$$\binom{d_a}{2} = \frac{d_a(d_a - 1)}{2}, \quad (2)$$

edges can exist, so the additional block fill-ins caused by eliminating  $a$  is

$$\Delta f_a = \binom{d_a}{2} - e_a = \frac{d_a(d_a - 1)}{2} - e_a. \quad (3)$$

Minimising fill-ins is turned into a reinforcement-learning objective by using

$$r_a = -\Delta f_a, \quad (4)$$

as the immediate reward, and

$$R_{\text{terminal}} = - \sum_{a \in \text{sequence}} \Delta f_a. \quad (5)$$

as the terminal reward. Thus the agent receives feedback proportional to the block fill-ins introduced at each step and is guided toward a globally efficient, block-aware elimination order.

### C. Multi-Relational GNN Feature Extractor

We adopt a two-layer GraphSAGE network to compute node embeddings on the block graph. Each node represents a block situated on the main diagonal of the sparse matrix—the atomic unit that may be eliminated during LU factorization. Its initial feature vector

$$\mathbf{x}_v = [m_v, d_v, d_D(v), d_F(v), \tilde{f}_v, \rho_v]^\top$$

collects both static and dynamic cues:  $m_v \in \{0, 1\}$  is a mask bit indicating whether the block is still eligible for elimination;  $d_v = |N(v)|$  is the current total degree;  $d_D(v) = |N_D(v)|$  and  $d_F(v) = |N_F(v)|$  count neighbours connected through **D**-type (band-diagonal) and **F**-type (full) edges, respectively;  $\tilde{f}_v = \binom{|N(v)|}{2} - |E(N(v))|$  estimates how many fill-in blocks would be created if  $v$  were removed; and  $\rho_v = \frac{|E(N(v))|}{\binom{|N(v)|}{2}}$  measures neighbour density, reflecting the residual risk of further fill-ins.

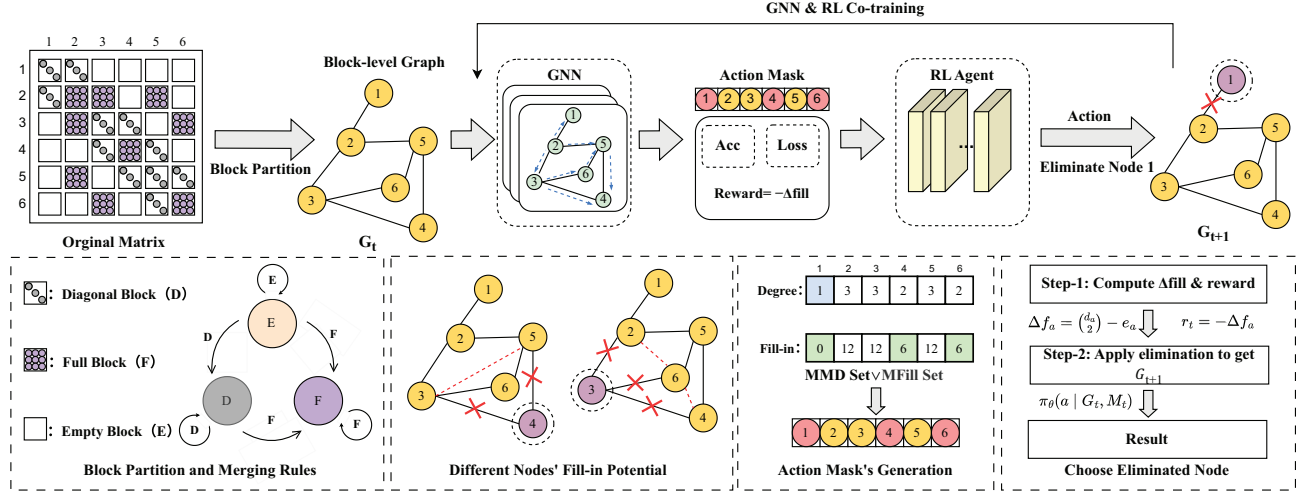


Fig. 3: Overall framework of MinFill algorithm.

Edges inherit the matrix’s intrinsic EDF pattern: an **F** block yields an F-type edge  $(v_i, v_j)$ , a **D** block yields a D-type edge, and an **E** block yields none. Runtime fill-in edges may either be merged into the F class or stored as a separate relation, provided the implementation matches the description.

At layer  $l$ , node  $v$  is updated by relation-specific mean aggregation:

$$\begin{aligned}
 h_v^{(l+1)} = & \sigma\left(W_s^{(l)} h_v^{(l)}\right) \\
 & + W_F^{(l)} \frac{1}{|N_F(v)|} \sum_{u \in N_F(v)} h_u^{(l)} \\
 & + W_D^{(l)} \frac{1}{|N_D(v)|} \sum_{u \in N_D(v)} h_u^{(l)}.
 \end{aligned} \quad (6)$$

where  $h_v^{(l)}$  is the embedding at layer  $l$ ,  $W_s^{(l)}$ ,  $W_F^{(l)}$ ,  $W_D^{(l)}$  are learnable weight matrices, and  $\sigma(\cdot)$  is a non-linear activation (ReLU in our experiments). Stacking  $L$  such layers produces  $h_v^{(L)}$ , a high-level representation fusing local structure and dynamic elimination status. The resulting embeddings are fed into the PPO policy head, which scores the priority of each block and samples the next elimination action under the guidance of an identical action mask.

#### D. Masked PPO Algorithm with $\varepsilon$ -Greedy Exploration

In our reinforcement learning framework, the elimination process on the EDF block graph is modeled as a Markov Decision Process (MDP). The state  $s$  consists of the current block graph, where nodes represent the remaining (non-eliminated) diagonal blocks and edges reflect the EDF structure. The action  $a$  corresponds to selecting one non-eliminated node for elimination. The state transition is defined by the fill-in update rule: when a node is eliminated, the environment removes it and its associated edges from the graph, and adds new edges between its neighbors if a previously empty block becomes a

full block. Each elimination step yields an immediate reward  $r$ , defined as the negative of the fill-ins generated. An episode terminates when all nodes have been eliminated, so that the cumulative reward corresponds to the negative total fill-ins. Maximizing the cumulative reward is equivalent to minimizing the overall fill.

Because the initial action space is large, with  $N$  candidate nodes, searching over all nodes is inefficient and prone to combinatorial explosion. Therefore, we introduce an action mask to restrict the set of allowable actions in each state to only the most promising candidates. Specifically, for every remaining node, we calculate its degree (i.e., the number of neighbors) and its fill-in potential. We then form the candidate action set as the union of the nodes with the minimum degree and those with the minimum fill-in potential. Define the binary mask  $M(s)$  as follows:

$$M(s)_i = \begin{cases} 1, & \text{if node } i \text{ is in the candidate set,} \\ 0, & \text{otherwise.} \end{cases}$$

The masked policy is computed by renormalizing the original policy output  $\pi(a|s)$ :

$$\tilde{\pi}(a|s) = \frac{\pi(a|s) M(s)_a}{\sum_{a'} \pi(a'|s) M(s)_{a'}}. \quad (7)$$

Although the action mask efficiently narrows the search to high-quality candidates by combining the MMD with the Minimum Fill-in Heuristic, a strict mask may exclude actions that could lead to better global solutions. To address this, we introduce an  $\varepsilon$ -greedy exploration mechanism: at each step, with probability  $\varepsilon$ , the agent temporarily relaxes the mask and samples a random action from the broader space, including nodes outside the candidate set. With probability  $1 - \varepsilon$ , it selects according to the masked distribution. This ensures that most decisions follow the heuristic candidates, while occasional exploration of unconventional actions remains possible.

We optimize the masked policy with PPO, using a GNN-based Actor for valid actions and a Critic sharing the GNN representation with global pooling. Trajectories of states, actions, and rewards are collected, and outputs are renormalized by the action mask before computing the PPO loss, so that only valid actions affect the update.

Combining the mask with  $\varepsilon$ -greedy exploration reduces the branching factor, focuses learning on near-optimal candidates, and occasionally explores better actions, accelerating convergence and avoiding suboptimal policies.

**Algorithm 1** PPO Training with Block-Level Action Mask

```

1: Init policy  $\theta$ , value  $\phi$ 
2: for  $episode = 1$  to  $N$  do
3:    $s \leftarrow \text{RESET}()$ ,  $\mathcal{T} \leftarrow \emptyset$ 
4:   while  $s$  not terminal do
5:      $\pi \leftarrow \text{POLICY}(s)$ ,  $M \leftarrow \text{MASK}(s)$ 
6:      $\tilde{\pi} \propto \pi \odot M$ ;  $a \sim \varepsilon\text{-greedy}(\tilde{\pi})$ 
7:      $(s', r) \leftarrow \text{STEP}(a)$ ;  $\mathcal{T} \cup = (s, a, r)$ ;  $s \leftarrow s'$ 
8:   end while
9:   Compute advantages  $A$  and returns  $G$  from  $\mathcal{T}$ 
10:  Update  $\theta, \phi$  with PPO objective using  $(\mathcal{T}, A, G)$ 
11: end for

```

IV. EXPERIMENTAL RESULTS

A. Experimental Setup

TABLE I: Information of 6 RF circuit matrices.

Datasets	Size	#NNZ
RF_1	5810K×5810K	194M
RF_2	419K×419K	922M
RF_3	2010K×2010K	106M
RF_4	3120K×3120K	243M
RF_5	1240K×1240K	26M
RF_6	44K×44K	2M

To preserve realistic RF sparsity while keeping training affordable, we use the largest industrial matrix RF\_1 (5.81M×5.81M) with diverse E/D/F blocks as the sparsity-pattern source. Using a 64×64 sliding window, we extract 100 local blocks (order 100–500) to construct 100 synthetic EDF matrices (80 for training and 20 for validation). Training is conducted only on these synthetic matrices; no full industrial matrix is used for training. We evaluate on six full industrial RF circuit matrices (RF\_1–RF\_6) ranging from 44K to 5.81M rows (Table I). For scalability, we first apply METIS *ndmetis* to build an elimination tree, then run MinFill-RL on each leaf subgraph (300–900 rows) and concatenate the resulting permutations. MinFill is implemented in Python 3.10/PyTorch 2.1 and evaluated with SuperLU\_DIST 9.0.0 on an AMD EPYC 7702 CPU (2.0GHz), 512GB RAM, and an RTX 4070 GPU. Baselines include SuiteSparse COLAMD, METIS *ndmetis*, and two block-level reordering methods. All methods are compared under identical hardware and solver

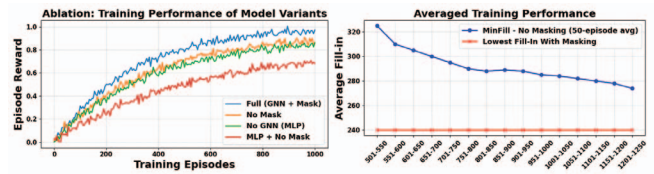


Fig. 4: Average training reward over five seeds for the full model and three ablated variants.

settings in terms of fill-ins, memory usage, reordering time, and factorization time.

B. Training Convergence and Robustness

Fig. 4 reports the mean episode reward over five random seeds. The full model (MinFill) converges rapidly, saturating near 0.92 within 800 episodes. Ablation studies reveal the contribution of each component: replacing the GNN with an MLP (No GNN (MLP)) results in a noticeable performance drop, indicating the importance of graph structural features. However, removing the action mask (No Mask) leads to an even more severe degradation, underscoring that the block-level mask is critical for constraining the search space to valid, high-quality candidates. Disabling both components yields the worst performance, with the reward stabilizing near 0.60. The right subfigure further validates this in terms of fill-in counts: the masked policy consistently achieves the lowest fill-ins, whereas the unmasked approach converges slowly and settles at a significantly higher fill-in level.

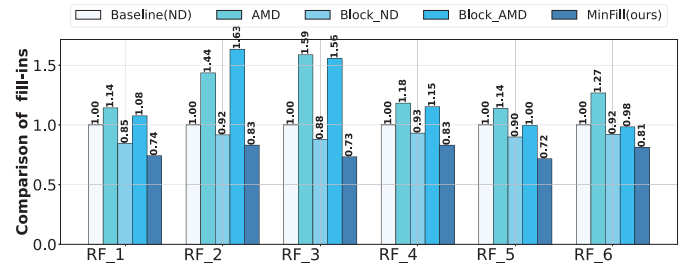


Fig. 5: The number of fill-ins comparison of all reordering algorithms.

C. Comparisons about the Number of Fill-ins

We first evaluate MinFill’s performance in reducing fill-ins by comparing it with ND [7], AMD [6], Block\_ND, and Block\_AMD. As illustrated in Fig. 5, MinFill consistently shows a significant reduction in fill-ins across all tested datasets. On average, MinFill achieves a 22.2% reduction compared with ND, a 38.9% reduction compared with AMD [6], and reductions of approximately 17.5% and 30.2% compared with Block\_ND and Block\_AMD, respectively. These results underscore MinFill’s superior capability to minimize fill-ins generation, particularly for matrices with multi-scale properties and locally dense blocks.

MinFill also delivers notable improvements on individual datasets. For example, on RF\_5, it achieves a 27.8% reduction in fill-ins compared with ND [7] and a 20.3% reduction compared with Block\_ND. On RF\_3, MinFill reduces fill-ins

TABLE II: Factorization time (s) and MinFill speedups over ND and AMD.

Dataset	ND [7]	AMD [6]	B_ND	B_AMD	MinFill Time (s)	Speed-up ND / AMD
RF_1	831.3	920.1	796.7	904.1	740.3	1.12× / 1.24×
RF_2	208.4	286.5	192.8	298.2	164.1	1.27× / 1.75×
RF_3	149.1	217.5	127.9	214.0	102.4	1.46× / 2.12×
RF_4	192.3	244.1	183.7	227.2	167.9	1.14× / 1.45×
RF_5	20.2	24.7	18.7	20.2	18.1	1.11× / 1.36×
RF_6	4.3	4.7	4.1	4.2	3.8	1.13× / 1.24×
Average	-	-	-	-	-	1.21× / 1.53×

by 50.4% relative to AMD [6] and by 40.5% compared with Block\_AMD. These gains mainly stem from MinFill’s fill-in potential-driven candidate selection strategy, which integrates factors such as node connectivity, block density, and full-block connectivity to accurately predict and reduce fill-ins generation.

#### D. Comparisons about the Memory Usage

Fig. 6 compares the memory usage of different reordering methods. We use ND [7] as the baseline, and the y-axis reports the memory cost of each method normalized to ND (lower is better). Across all test datasets, MinFill achieves lower memory cost than ND [7], AMD [6], and Block\_AMD. On average, MinFill reduces memory cost by approximately 12.2% relative to ND, 28.6% relative to AMD, and 26.7% relative to Block\_AMD. The reduction relative to Block\_ND is much smaller (0.36% on average), because Block\_ND applies block-level preprocessing that aggregates each block as a single unit in the sparsity pattern, which already decreases the factor memory cost by construction and leaves limited headroom for further reduction. Even so, MinFill matches or slightly improves upon Block\_ND on most datasets while maintaining clear advantages over the other baselines.

#### E. Comparisons about the Factorization Time

Table II reports the factorization time (s) of all methods and the speedups of MinFill over ND [7] and AMD [6], which are also visualized in Fig. 7. The speedup curves in Fig. 7 are consistent with the timing results in Table II, showing that MinFill delivers stable gains across matrices of different sizes. MinFill achieves the lowest factorization time on all six RF matrices, delivering 1.11×–1.46× speedups over ND and 1.24×–2.12× over AMD (1.21×/1.53× on average). Moreover, MinFill also consistently outperforms the two block-aware baselines (Block\_ND and Block\_AMD), indicating that the learned block-level elimination policy can complement preprocessing-based block aggregation. These improvements mainly come from reducing fill-ins, which lowers both the amount of numerical work and the memory traffic during factorization, thereby accelerating the overall solve.

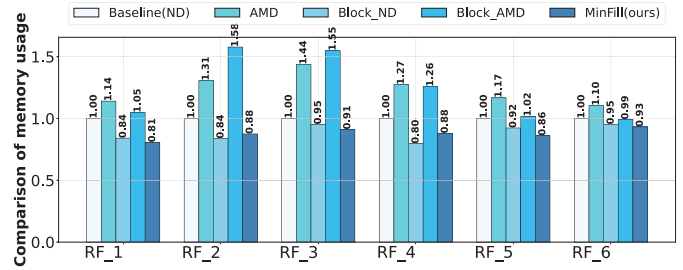


Fig. 6: Memory usage comparison of all reordering algorithms.

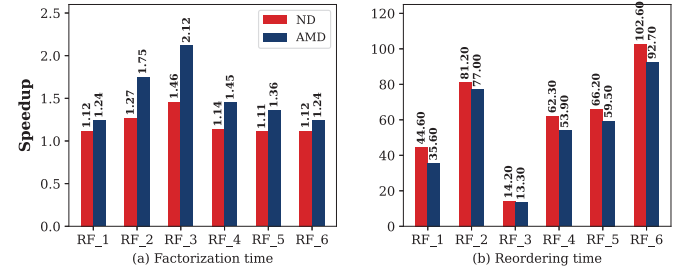


Fig. 7: Speedup of MinFill relative to ND and AMD for (a) factorization time and (b) reordering time.

#### F. Comparisons about the Reordering Time

Fig. 7 shows that our proposed MinFill algorithm achieves substantial speedups in reordering time compared to traditional algorithms. Specifically, MinFill exhibits an average speedup of approximately 61.9× over the ND [7] algorithm and 55.4× over the AMD [6] algorithm across all test datasets. This highlights a significant reduction in reordering time. Moreover, when considering the total runtime, which includes both reordering and factorization, MinFill consistently demonstrates superior overall performance across all datasets.

## V. CONCLUSION

This paper presents MinFill, a block-aware sparse-matrix reordering framework for large RF simulations. By abstracting the matrix into a tri-relational graph of empty, diagonal, and full blocks, feeding them to a multi-relational GNN, and learning elimination policies with a maskable PPO with an incremental-fill penalty, MinFill produces sparsity-preserving orders without hand-tuned heuristics. Experiments on six RF matrices show MinFill reduces fill-ins by 22% vs ND and 39% vs AMD, lowers peak memory by up to 29%, shortens factorization time by one-third, and accelerates reordering by over 60×. These results confirm that exploiting block structure enables an effective and scalable solution for large-scale RF simulation.

## VI. ACKNOWLEDGMENT

This work was supported by NSFC (Grant No. 92473107, 12526211, 62234010, 62374031), Jiangsu Province NSF (Grant No. BK20240173).

## REFERENCES

- [1] T. A. Davis and E. P. Natarajan, "Sparse matrix methods for circuit simulation problems," in *Scientific Computing in Electrical Engineering (SCEE)*, 2010.
- [2] A. Pothen and C. Fan, "Computing the block triangular form of a sparse matrix," *ACM Transactions on Mathematical Software*, 1990.
- [3] A. Mehrotra and A. Somani, "A robust and efficient harmonic balance (HB) using direct solution of HB jacobian," in *Proceedings of the 46th Annual Design Automation Conference*, 2009.
- [4] L. Han, X. Zhao, and Z. Feng, "An adaptive graph sparsification approach to scalable harmonic balance analysis of strongly nonlinear post-layout RF circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2014.
- [5] H. M. Markowitz, "The elimination form of the inverse and its application to linear programming," *Management Science*, 1957.
- [6] P. R. Amestoy, T. A. Davis, and I. S. Duff, "Algorithm 837: AMD, an approximate minimum degree ordering algorithm," *ACM Transactions on Mathematical Software*, 2004.
- [7] A. George, "Nested dissection of a regular finite element mesh," *SIAM Journal on Numerical Analysis*, 1973.
- [8] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.
- [9] M. Schlichtkrull, T. N. Kipf, and P. Bloem, "Modeling relational data with graph convolutional networks," in *The Semantic Web: 15th International Conference (ESWC)*, 2018.
- [10] P. Veličković, G. Cucurull, and A. Casanova, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.
- [11] J. Gasteiger, J. Groß, and S. Günnemann, "Directional message passing for molecular graphs," *arXiv preprint arXiv:2003.03123*, 2020.
- [12] A. Gatti, Z. Hu, T. Smidt, E. G. Ng, and P. Ghysels, "Graph partitioning and sparse matrix ordering using reinforcement learning and graph neural networks," *Journal of Machine Learning Research*, 2022.
- [13] A. Dasgupta and P. Kumar, "Alpha elimination: Using deep reinforcement learning to reduce fill-in during sparse matrix decomposition," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, 2023.
- [14] E. Harb and H. S. Lam, "Refill: Reinforcement learning for fill-in minimization," *arXiv preprint arXiv:2501.16130*, 2025.
- [15] A. Raffin, A. Hill, and A. Gleave, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, 2021.
- [16] X. S. Li and J. W. Demmel, "SuperLU\_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems," *ACM Transactions on Mathematical Software*, 2003.
- [17] O. Schenk and K. Gärtner, "Solving unsymmetric sparse systems of linear equations with PARDISO," *Future Generation Computer Systems*, 2004.
- [18] X. Fu, B. Zhang, T. Wang, W. Li, Y. Lu, E. Yi, J. Zhao, X. Geng, F. Li, J. Zhang *et al.*, "Pangulu: A scalable regular two-dimensional block-cyclic sparse direct solver on distributed heterogeneous systems," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2023.
- [19] C. W. Bomhof and H. A. van der Vorst, "A parallel linear system solver for circuit simulation problems," *Numerical Linear Algebra with Applications*, 2000.
- [20] D. Niu, Y. Tao, Z. Jin, Y. Dong, C. Wang, and C. Sun, "Islu: Indexing-efficient sparse lu factorization for circuit simulation on gpus," in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, 2024.
- [21] X. Chen, Y. Wang, and H. Yang, "NICS LU: An adaptive sparse matrix solver for parallel circuit simulation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2013.
- [22] T. A. Davis and E. Palamadai Natarajan, "Algorithm 907: KLU, a direct sparse solver for circuit simulation problems," *ACM Transactions on Mathematical Software*, 2010.
- [23] J. Zhao, Y. Wen, Y. Luo, Z. Jin, W. Liu, and Z. Zhou, "Sflu: Synchronization-free sparse lu factorization for fast circuit simulation on gpus," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*, 2021.
- [24] G. Feng, H. Wang, Z. Guo, M. Li, T. Zhao, Z. Jin, W. Jia, G. Tan, and N. Sun, "Accelerating large-scale sparse lu factorization for rf circuit simulation," in *European Conference on Parallel Processing*, 2024.
- [25] —, "Efficient large-scale sparse lu factorization for fast radio frequency circuit simulation," *The International Journal of High Performance Computing Applications*, 2025.
- [26] A. George and J. W. H. Liu, "The evolution of the minimum degree ordering algorithm," *SIAM Review*, 1989.
- [27] J. W. H. Liu, "Modification of the minimum-degree algorithm by multiple elimination," *ACM Transactions on Mathematical Software*, 1985.
- [28] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on Scientific Computing*, 1998.
- [29] C. Chevalier and F. Pellegrini, "PT-Scotch: A tool for efficient parallel graph ordering," *Parallel Computing*, 2008.
- [30] P. Sanders and C. Schulz, "Engineering multilevel graph partitioning algorithms," in *European Symposium on Algorithms*, 2011.
- [31] S. W. Sloan, "An algorithm for profile and wavefront reduction of sparse matrices," *International Journal for Numerical Methods in Engineering*, 1986.
- [32] N. E. Gibbs, W. G. Poole, Jr, and P. K. Stockmeyer, "An algorithm for reducing the bandwidth and profile of a sparse matrix," *SIAM Journal on Numerical Analysis*, 1976.
- [33] J. D. Trotter, S. Ekmekçi, J. Langguth, T. Torun, E. Düzakın, A. Ilic, and D. Unat, "Bringing order to sparsity: A sparse matrix reordering study on multicore cpus," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2023.
- [34] Y. Lu, Y. Luo, H. Lian, Z. Jin, and W. Liu, "Implementing lu and cholesky factorizations on artificial intelligence accelerators," *CCF Transactions on High Performance Computing*, 2021.
- [35] C. B. Browne, E. Powley, and D. Whitehouse, "A survey of monte carlo tree search methods," *IEEE Transactions on Computational Intelligence and AI in Games*, 2012.
- [36] R. Xia, J. Guo, and H. Zhang, "Sparse matrix reordering method selection with parallel computing and deep learning," in *2024 International Joint Conference on Neural Networks (IJCNN)*, 2024.