GNN-Guided Graph Partitioning for Spatial CGRA Mapping

Deniz Lopes Gunes[1][0009−0009−7712−1808], Henrique Teixeira[1][0009−0000−6764−0449], Guilherme Oliveira[1][0009−0007−3352−4344], and Nuno Paulino[1][0000−0001−5547−0323]

INESC TEC, Porto, Portugal
{deniz.l.gunes,henrique.santos.teixeira,
guilherme.oliveira,nuno.m.paulino}@inesctec.pt

Abstract. Mapping Data-Flow Graphs (DFGs) onto Coarse-Grained Reconfigurable Arrays (CGRAs) requires partitioning the graph into resource-feasible subgraphs when the DFG exceeds CGRA capacity. The quality of this partitioning directly impacts execution efficiency: each cut edge incurs communication overhead due to data transfer. Exact methods such as exhaustive SAT-based min-K partitioning yield optimal cuts but scale poorly; greedy heuristics are fast but produce poor partitions and frequently fail on wide, parallel graphs. This paper proposes a GNN-guided partitioning flow that learns optimal graph partitions, by learning from SAT-generated ground truth. The GNN predicts per-node partition assignments, which seed a topology-aware partitioner that convexifies dependency ordering and repairs capacity violations to guarantee feasibility. On a synthetic DFG dataset with 1650 graphs (150 held-out for evaluation), the GNN-guided partitioner reduces edge cuts by $\sim$1.07 per graph relative to the unguided baseline ($\approx$33% relative reduction, roughly halving the gap to the SAT optimum), maps 98% of graphs versus 15% for greedy list scheduling, and runs at $\sim$0.25 s per graph; approximately 50$\times$ faster than the exact SAT solver.

Keywords: Coarse-Grained Reconfigurable Arrays · Graph Partitioning · Graph Neural Networks · Spatial Mapping · Data-Flow Graphs · SAT-Based Mapping

1 Introduction

Coarse-Grained Reconfigurable Arrays (CGRAs) have been a topic of hardware architecture research for three decades, and despite extant limitations, they are the target of newfound interest due to emergence of Edge AI applications, such as Small Language Models or multi-modal transformers. The bulk of the processing in these workloads is based on matrix multiplication, and thus the spatial parallelism of CGRAs promises processing benefits, especially when considering embedded and edge scenarios which can also benefit from the open-source RISC-V ecosystem for accelerated integration and cost-free IP use.

However, mapping computation, which is typically in the form of Data-Flow Graphs (DFGs), onto CGRAs remains an NP-Hard problem. It is a placement and routing problem which becomes computationally very expensive, as the size of the CGRA or the DFG increases. Furthermore, while approaches differ, DFGs must typically be partitioned temporally to achieve total mapping onto the target CGRAs resource constraints, resulting in multiple execution steps. Since data must be transferred between stages, the quality of graph partitioning affects routing process as well as execution time. Approaches that guarantee the highest quality partition such as an exhaustive search require a large computational cost, having to map multiple possible partitions to compare their mapping quality. Naive or greedy approaches for partitioning reduce the computational cost at the cost of poorer quality partitions with higher edge-cut counts. In contrast, recent works such as LISA [5] and GRAFT [4] explore the use of Graph Neural Networks (GNNs) for CGRA mapping, targeting placement prediction and routing guidance.

This work likewise adopts a GNN based approach, but focuses on learning the best graph partitions prior to mapping. The approach generates ground-truth labels through an exhaustive SAT-min-k (Algorithm 1) partitioning strategy and uses them to train a GNN capable of predicting partition placements for unseen DFGs. During inference, per-node partition hints seed a topology-aware partitioner that guarantees feasibility while preserving the learned partition structure.

2 Related Work

Mapping a DFG onto a CGRA is a well-known NP-hard problem, involving simultaneous placement and routing under resource constraints. Exact formulations such as Integer Linear Programming (ILP) and Boolean Satisfiability (SAT) guarantee optimal solutions but scale poorly, since as DFG and CGRA sizes grow, the search space expands exponentially. At the other extreme, heuristic mappers such as DRESC [6] use simulated annealing to solve placement, scheduling and routing simultaneously, trading optimality guarantees for tractable compile time on larger graphs. SAT-based approaches such as Miyasaka et al. [7] encode the mapping constraints directly as a Boolean formula, while SAT-MapIt [8] narrows the search space by computing a Kernel Mobility Schedule and then applying SAT to find an optimal modulo schedule, but remain targeted at the single-configuration placement problem rather than multi-partition spatial mapping.

When a DFG exceeds the physical capacity or routing capability of the target CGRA, the graph must be partitioned into multiple temporal or spatial subgraphs. Traditional approaches apply generic graph-partitioning algorithms such as the Kernighan-Lin heuristic or multi-level frameworks such as METIS [2] to minimize edge cuts. These methods are agnostic to CGRA topology and routing constraints, so the partitions they produce are not guaranteed to be embeddable onto the target hardware. Greedy heuristics reduce computational cost further but frequently fail on wide, parallel graphs and produce high cut counts

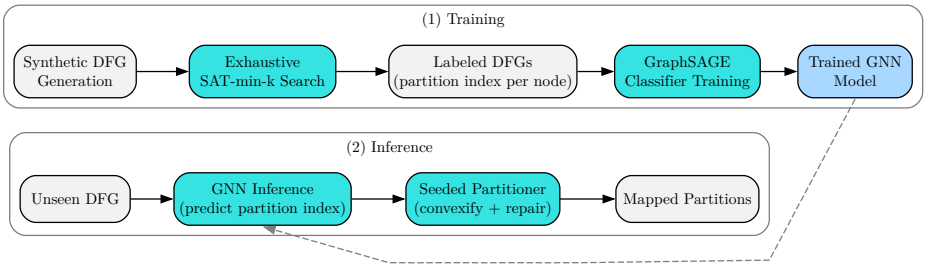


Fig. 1: Overview of the proposed flow. The methodology is divided into two distinct phases: training the GNN and inference-time graph partitioning with SAT-based spatial validation

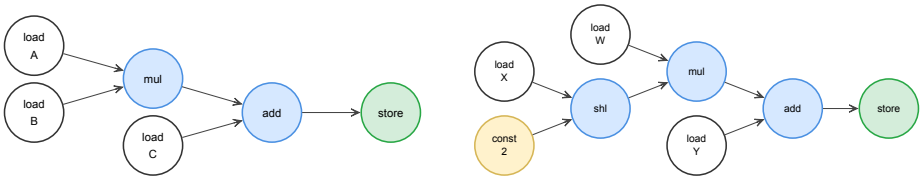
when they do succeed, each cut edge introducing communication overhead in the form of a synthetic memory operation. Closer to our setting, Kojima et al. [3] apply deep learning to make kernel partitioning mapping-aware, steering the partitioner toward sub-DFGs that are more likely to map successfully. Unlike our approach, the learned model assesses partition mappability rather than directly guiding topology-feasible cuts with SAT-derived supervision.

Recent work has demonstrated that GNNs can accelerate the downstream mapping process by predicting labels that reduce the search space for iterative mappers. LISA [5] trains GNNs to predict per-node scheduling, placement, and mapping-distance labels that guide a simulated annealing mapper, reporting up to 724× faster compilation than ILP and 17× over vanilla simulated annealing while mapping far more benchmarks. GRAFT [4] instead predicts a placement distribution through GNN-based subgraph matching to steer a progressive graph-based mapper, reporting a 64.5× mapping speedup over LISA with comparable or better mapping quality on CGRAs up to 16×16. Both approaches assume the DFG fits within a single CGRA configuration and focus on accelerating placement and routing; neither addresses the partitioning problem that arises when this assumption does not hold.

3 Proposed Approach

The primary goal of our approach is to achieve the high partition quality of exact mathematical solvers while maintaining the low computational overhead of greedy heuristics. As shown in Fig. 1, the proposed flow comprises three stages: (1) Synthetic Graph Generation, (2) Ground-Truth Generation and (3) GNN-Guided Partitioning and Spatial Mapping. Steps (1) and (2) constitute the phases of GNN training and (3) represents DFG partitioning inference.

In summary, we train a GNN which predicts, for each instruction node of an input DFG, the partition it should belong to in an optimal split. This split is one which minimizes the inter-partition cut edges while maintaining spatial mapability on the target CGRA. To obtain a ground-truth relating mapping quality



(a) Matmul kernel template (b) Convolution kernel template

Fig. 2: Kernel templates used as a base for DFG generation.

to graph splits, in order to feed the GNN training, we perform an exhaustive SAT-MIN-K (Algorithm 1) per training graph. That is, we attempt to map every possible graph split for per training DFG and record the mapping quality versus the chosen split.

The trained GNN then predicts, for an input test DFG, the best partition to which each node should belong, and adds this information as attributes to the nodes. A topology-aware partitioner then processes the annotated graph, performing a final refinement to ensure mapping validity given the CGRAs hardware constraints. Thus, by reducing the mapping search space to an initial inferred solution, the DFG mapping time is reduced.

The following sections detail the synthetic DFG generation, the utilized SAT mapper for ground-truth labeling, the GNN training process, and the final process for GNN-guided partitioning and mapping.

3.1 Synthetic Graph Generation

Training a GNN-based mapper requires a large corpus of labeled DFGs that span a representative range of kernel shapes, data-dependency structures, and memory-access patterns, while remaining within the resource bounds of the target CGRA. Because hand-written or compiler extracted benchmarks are too few and too uniform for this purpose, we built a parametric generator that produces synthetic DFGs by starting from a small kernel template and applying a sequence of structure-preserving mutations.

The two loop-body templates shown in Fig. 2 seed the generator. The *matrix-multiplication* template encodes the inner accumulation $c \mathrel{+}= a[k] \times b[k]$ as a six-node DAG: two loads feeding a *mul*, whose result is added to a third load and then written back by a *store*. The *convolution* template encodes $y \mathrel{+}= w[k] \times (x[k] \ll 2)$ and extends this chain with a constant node and a *shl* operation, yielding an eight-node seed. Both templates are acyclic, representing one inner loop iteration, and have exactly one memory write, and satisfy all CGRA structural invariants by construction. The types of nodes are drawn from a set of ten operations: arithmetic (*add*, *sub*,*mul*), bitwise logic (*and*, *or*, *xor*), shifts (*shl*, *shr*), and memory operations (*load*, *store*). Immediate constants are modeled as dedicated constant nodes with a value sampled uniformly from $[-256, 256]$. To increase

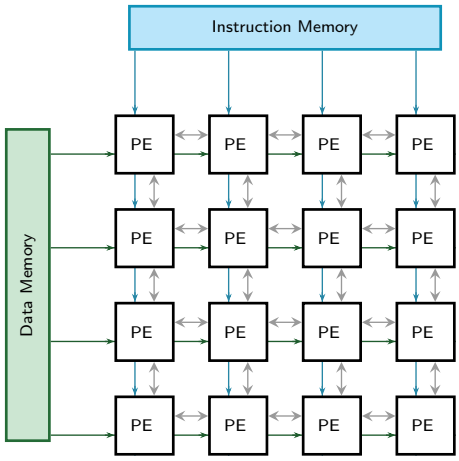


Fig. 3: Targeted 4×4 Homogeneous CGRA; 16 PE's with full ALU capability and Memory access, nearest neighbour connection.

graph variability the generator applies a sequence of graph mutations over the templates. Mutations include three structural types:

- *Depth mutations* lengthen the critical path by inserting operations along existing producer–consumer chains, increasing pipeline depth.
- *Breadth mutations* introduce new parallel computation by branching live values into independent paths, merging independent values through binary operations, or adding fan-out to existing nodes without extending depth. These mutations increase ASAP-level width and routing pressure.
- *Neutral mutations* alter operations or operands types without changing graph topology, replacing nodes with others of the same type, or replacing a memory load with an immediate constant operand. These mutations increase operation-type diversity without affecting the graph's structural complexity.

3.2 SAT-based Spatial Mapper

To place and route a DFG that fits on the target CGRA (Fig. 3), we use a custom SAT placement encoding, distinct from the modulo-scheduling formulation of SAT-MapIt [8], that assigns nodes to PEs subject to the following conditions:

Let V_{I} be the set of instruction nodes, E_{I} the set of instruction-to-instruction edges, and $\mathcal{Q}$ the set of PEs of the CGRA. For every $(v, q) \in V_{\mathrm{I}} \times \mathcal{Q}$ we introduce a Boolean placement variable

$$x_{v,q} \in \{0, 1\}, \qquad x_{v,q} = 1 \iff \text{node } v \text{ is placed on PE } q.$$

The mapping is then encoded by the following clauses:

(C1) Operation compatibility. Let $\mathrm{op}(v)$ denote the opcode of node v and $\mathrm{ops}(q)$ the set of opcodes supported by PE q:

$$\forall v \in V_{\mathrm{I}},\ \forall q \in \mathcal{Q},\ \mathrm{op}(v) \notin \mathrm{ops}(q) \implies \neg x_{v,q}. \tag{1}$$

(C2, C3) One-to-one placement. Each instruction occupies exactly one PE, and each PE holds at most one instruction:

$$\forall v \in V_{\mathrm{I}},\ \sum_{q \in \mathcal{Q}} x_{v,q} = 1 \qquad \text{and} \qquad \forall q \in \mathcal{Q},\ \sum_{v \in V_{\mathrm{I}}} x_{v,q} \leq 1. \tag{2}$$

(C4) Same-cycle nearest-neighbor dependencies. Let $\mathrm{nbr}(q, q')$ denote that PEs q and q' are directly connected in the CGRA topology. For every dependency $u \to v \in E_{\mathrm{I}}$:

$$\forall (u, v) \in E_{\mathrm{I}},\quad \bigvee_{\substack{q,q' \in \mathcal{Q} \\ \mathrm{nbr}(q,q')}} \big(x_{u,q} \wedge x_{v,q'}\big). \tag{3}$$

3.3 Ground-Truth Generation and GNN Training

After generating a set of DFGs, we build a training dataset for the GNN by systematic partition and mapping to valid solutions. Specifically, we define a valid solution by partitioning the DFG nodes into K non-empty subsets. When edges are cut between these subsets, they are transformed into synthetic *load*/*store* pairs; each resulting subset must then successfully mapped.

An edge that crosses a partition boundary cannot be represented as a direct connection between PEs, because the two operations belong to distinct configurations. For a non-load producer, the cut edge $u \to v$ is represented as a *store* at the end of P_i and a *load* at the start of P_j.

The objective of generating ground-truth is to find the partition assignment σ^* that jointly minimizes the number of partitions K and the number of cut edges, subject to CGRA constraints. Once σ^* is found, each instruction node $v \in V_I$ is labeled with its assigned partition index $\sigma^*(v)$, which serves as the ground-truth target for the GNN's multi-class classification.

Specifically, we perform an exhaustive SAT-MIN-K (Algorithm 1) search over each synthetic DFG. The algorithm performs an upper-bounded sweep over K, starting from a resource-derived lower-bound $K_{\min}$(the instruction count $|V_I|$ divided by the number of PEs) up to $|V_I|$, the trivial case in which each partition contains a single instruction. Because partitions execute sequentially, a value can only be read from an equal-or-earlier partition, so a producer's partition index never exceeds its consumer's. That is, only *monotone* assignments, which respect the topological order of G, are considered. For each value of K, all valid monotone assignments are enumerated exhaustively and SAT-checked, guaranteeing that the returned partition is minimal in K and optimal in edge cuts among all feasible K-partition solutions. Each feasible partition is scored by the tuple:

$$\mathcal{S}(\mathcal{P}) = \big(\,|\mathrm{cut}(\mathcal{P})|,\ |V_{\mathrm{syn}}|,\ \max_i f_i,\ \max_i \rho_i,\ \big) \tag{4}$$

where $|\mathrm{cut}(\mathcal{P})|$ is the number of cut edges, $|V_{\mathrm{syn}}|$ the number of data transfers introduced by partitioning, $\max_i f_i$ the maximum frontier edge count across partitions, $\max_i \rho_i$ the maximum PE occupancy. Candidates are ranked lexicographically by $\mathcal{S}$, so cut minimization is the primary objective and the remaining terms act as tiebreakers in order of priority. Among all feasible K-partition candidates, the one minimizing $\mathcal{S}$ is selected as the ground-truth solution for that graph.

Algorithm 1 SAT-MIN-K Partitioning

Require: DFG $G = (V, E)$, CGRA architecture $\mathcal{A}$ ▷ $V_I \subseteq V$: instruction nodes
Ensure: Minimum-K feasible partition $\mathcal{P}^*$ with minimum edge cuts
  $K_{\min} \leftarrow \lceil |V_I| / \mathrm{capacity}(\mathcal{A}) \rceil$ ▷ Resource lower bound
  for $K \in \{K_{\min}, \ldots, |V_I|\}$ **do**
    for each monotone assignment $\sigma : V_I \to [K]$ of instruction nodes to K non-empty partitions **do**
      Prune σ if materialized resource usage is infeasible ▷ Early rejection
      Deduplicate σ against memoised isomorphic partitions ▷ Cache Hit
      Construct partitions $\mathcal{P} = \{P_1, \ldots, P_K\}$ from σ
      Materialize cut edges as synthetic *load*/*store* pairs
      if $\mathcal{P}$ is structurally infeasible (routing or capacity) **then**
        continue
      end if
      if SAT-BASED SPATIAL MAPPER(P_i, $\mathcal{A}$) is feasible $\forall i \in [K]$ **then**
        if $\mathcal{S}(\mathcal{P}) < \mathcal{S}(\mathcal{P}^*)$ **then**
          $\mathcal{P}^* \leftarrow \mathcal{P}$ ▷ lexicographic min, Eq. (4)
        end if
      end if
    end for
    if $\mathcal{P}^*$ is defined **then**
      return $\mathcal{P}^*$ ▷ Optimal K-partition; do not increment K
    end if
  end for

The ground-truth label assigned to each instruction node $v \in V_I$ is its partition index $\sigma^*(v) \in \{0, \ldots, K^* - 1\}$ under the selected assignment σ^*, where K^* is the minimum feasible K returned by SAT-MIN-K (Algorithm 1).

Each node is encoded with the feature set introduced by LISA [5]. The model, a multi-layer GraphSAGE [1] classifier, is trained with inverse-frequency-weighted cross-entropy loss to compensate for class imbalance arising from variable partition sizes across training graphs:

$$\mathcal{L} = -\sum_{v \in V_I} w_{\sigma^*(v)} \log \hat{p}\big(\sigma^*(v) \mid v\big) \tag{5}$$

where $\hat{p}(\cdot \mid v)$ is the softmax output of the classifier for node v and $w_k \propto 1/|\{v : \sigma^*(v) = k\}|$ is the inverse class frequency weight for partition k.

3.4 GNN-Guided Partitioning and Spatial Mapping

At inference time, the trained GNN assigns each node $v \in V_I$ of an unseen DFG an initial partition index $\hat{\sigma}(v)$ (its *seed*). These seeds do not guarantee directly mappable partitions, so a topology-aware partitioner refines them into a feasible one.

Feasibility. A partition P_i is *feasible* when its subgraph, including the synthetic *load*/*store* nodes introduced for cut edges, satisfies all of: 1) obeys checks on resource count and connectivity of CGRA, 2) number of edge cuts is lesser than specified budget, 3) is successfully mappable by SAT mapper. Otherwise, the *Repair* step is the procedure that turns the raw GNN seeding into a feasible partitioning by adjusting the partition indices and, where necessary, splitting partitions.

Partition repair. Rather than performing a conventional list-scheduling sweep, the partitioner takes all seeds $\hat{\sigma}$ at once and repairs them. It first makes the indices order-consistent: because partitions execute sequentially, a partition must be *convex* (topologically contiguous), i.e. no partition may hold a consumer placed before its producer. Each node is therefore pushed to at least one index beyond its latest predecessor:

$$\pi(v) = \max\left(\hat{\sigma}(v),\ \max_{(p \to v) \in E_I} \pi(p) + 1\right) \tag{6}$$

Memory loads, which have no predecessor, are then pulled forward to their earliest consumer so they share a partition with their first use:

$$\pi(\ell) = \min_{(\ell \to s) \in E_I} \pi(s) \qquad \forall\, \ell \in V_{\text{load}} \tag{7}$$

Convexification can leave empty indices; since an empty index would correspond to an empty, no-op configuration, the indices are compacted to a dense range $\{0, \ldots, K-1\}$, after which K is the actual number of partitions.

These partitions are checked and repaired in index order. A feasible partition is finalized immediately. An infeasible partition containing more than one node is split at its largest feasible topological prefix: the prefix is finalized, a synthetic *store*/*load* pair is materialized at the cut to represent the introduced cut edge, and the remaining suffix is re-enqueued at the front of the queue as a new candidate. Since a partition with a single node is always mappable, this converges to a feasible partitioning, at the cost of at most one additional cut edge per split.

After the repair step, each resulting partition P_i is mapped independently onto the CGRA by the SAT-based spatial mapper of Section 3.2, with one SAT solve per partition. Since every partition is capacity-feasible by construction, each solve operates on a subgraph that fits within the CGRA's spatial resources, keeping solve time tractable independently of the size of the original DFG.

Table 1: Structural statistics of the training (1500 graphs) and evaluation (150 graphs) dataset splits. All values reported as mean ± std (min–max).

Metric	**Training**	**Evaluation**
Nodes	22.51 ± 3.31 (16–28)	23.16 ± 2.92 (16–28)
Edges	21.77 ± 3.39 (15–29)	22.45 ± 2.97 (15–28)
Depth	8.19 ± 1.32 (6–13)	8.40 ± 1.30 (6–12)
Parallelism	4.26 ± 0.95 (2–8)	4.19 ± 0.97 (2–7)
Instr-Level Parallelism	2.21 ± 0.45 (1.3–4.0)	2.21 ± 0.48 (1.3–3.8)
Memory ratio	0.42 ± 0.07 (0.21–0.62)	0.42 ± 0.08 (0.25–0.60)

4 Experimental Evaluation

4.1 Experimental Setup

To evaluate the performance of the proposed GNN-guided partitioning framework a 4×4 homogeneous CGRA was targeted, shown in Fig. 3.

A dataset of 1650 DFGs was generated using the synthetic generator described in Section 3.1. Of these, 1500 graphs were used to produce ground-truth partition labels for GNN training, and the remaining 150 served as a held-out benchmark set for evaluating five mapping strategies.

Highly parallelizable graphs impose higher partition counts and routing constraints that are difficult to satisfy under the spatial mapper's nearest-neighbor adjacency requirement (C4). To relax routing pressure and keep training tractable, the training set was biased toward moderately sized, low-to-medium parallelism graphs. **Table 1** summarizes the structural statistics of both splits; the two distributions are closely matched, confirming that the evaluation set is representative of the training distribution.

Five partitioning strategies were evaluated to characterize the contribution of each component of the proposed framework. SAT-MIN-K (Algorithm 1) serves as the optimal reference, performing an exhaustive search over all valid K-partition assignments and returning the minimum-cut solution at the smallest feasible K. A greedy list-scheduling partitioner is included as a fast baseline, building partitions in a single irreversible sweep without searching over alternative assignments, representative of conventional compilation heuristics. To isolate the contribution of the guided partitioner independently of the GNN, the topological strategy runs the topology-aware partitioner with no learned input: all nodes are seeded to partition 0 and boundaries are determined entirely by the convexification and capacity repair steps (Section 3.4). The GNN-RAW strategy takes per-node partition indices predicted by the GNN and constructs partitions directly from these assignments without any convexification or feasibility repair, submitting them to the SAT-based spatial mapper as-is; this isolates the raw predictive quality of the GNN from the contribution of the repair procedure. Finally, the full GNN-GUIDED strategy seeds the topology-aware partitioner with GNN predictions, combining learned partition structure with topology-aware repair, and constitutes the primary contribution of this work.

Table 2: Benchmark results. (a) full 150-graph evaluation set; (b) restricted to the 106 graphs solved by GNN-RAW, the like-for-like subset for the GNN-RAW vs. GNN-GUIDED comparison. † Greedy results are reported on its successes only (23 in (a), 20 in (b)) and are not directly comparable.

(a) Full 150-graph evaluation set.

	SAT-MIN-K	Greedy	Topological	GNN-RAW	GNN-GUIDED
Success rate	100%	15.3%	92.7%	70.7%	98.0%
Mean edge cuts	1.31 ± 0.09	$1.52 \pm 0.37^{\dagger}$	3.22 ± 0.25	1.74 ± 0.17	2.15 ± 0.20
Partitioning (s)	0.581 ± 0.396	0.005 ± 0.001	0.007 ± 0.001	0.0004 ± 0.00001	0.004 ± 0.0004
SAT mapping (s)	11.508 ± 2.276	0.071 ± 0.004	0.498 ± 0.047	0.071 ± 0.002	0.243 ± 0.039
SAT checks / part.	80.59 ± 12.40	1.00 ± 0.00	3.71 ± 0.21	1.00 ± 0.00	2.36 ± 0.11

(b) 106-graph subset solved by GNN-RAW.

	SAT-MIN-K	Greedy	Topological	GNN-RAW	GNN-GUIDED
Success rate	100%	18.9%	94.3%	100%	100%
Mean edge cuts	1.09 ± 0.06	$1.55 \pm 0.41^{\dagger}$	3.00 ± 0.29	1.74 ± 0.17	1.60 ± 0.17
Partitioning (s)	0.089 ± 0.013	0.005 ± 0.001	0.005 ± 0.0005	0.0003 ± 0.00001	0.002 ± 0.0001
SAT mapping (s)	7.923 ± 1.265	0.061 ± 0.004	0.393 ± 0.044	0.060 ± 0.002	0.133 ± 0.028
SAT checks / part.	70.44 ± 8.87	1.00 ± 0.00	3.60 ± 0.24	1.00 ± 0.00	2.01 ± 0.02

Strategies. SAT-MIN-K: exhaustive search over partition assignments; optimal reference. Greedy: single-pass list scheduling, no repair (fast baseline). Topological: topology-aware partitioner with all nodes seeded to one partition; feasibility repair only. GNN-RAW: raw GNN predictions submitted directly, without feasibility checks or repair. GNN-GUIDED: topology-aware partitioner seeded with GNN predictions (proposed method).

The framework operates under two simplifying assumptions. Each partition is assumed to complete execution within a single CGRA configuration cycle, consistent with the one-shot mapping model of the SAT-based spatial mapper and a common assumption in spatial CGRA compilation; it does not account for multi-cycle operations or data-dependent latencies that would require a modulo scheduling pass. Cut edges are materialized as synthetic *store*/*load* pairs at partition boundaries, implying a shared data memory for transferring live values between consecutive configurations. The framework does not model transfer latency or bandwidth, nor architectures using dedicated inter-configuration registers or FIFOs. The number of cut edges serves as a proxy for inter-partition communication overhead under the assumption that each synthetic memory pair incurs a uniform cost.

4.2 Experimental Results

Feasibility. The greedy strategy succeeds on only 15.3% of graphs, confirming that single-pass list scheduling is not suited for wide, parallel DFGs. GNN-RAW, which submits raw GNN predictions directly to the SAT mapper without any

repair, achieves 70.7%, reflecting the current accuracy limitations of the GNN: predictions are occasionally structurally infeasible, but when they are valid they require no repair overhead. Both topology-aware variants achieve high feasibility rates: topological at 92.7% and GNN-GUIDED at 98.0%, demonstrating that the repair procedure alone largely solves the feasibility problem independently of the GNN.

Partition quality. SAT-MIN-K (Algorithm 1) achieves a mean of 1.31 edge cuts per graph, establishing the optimality baseline. The topological strategy produces 3.22 mean cuts, a gap of 1.91 above optimal, which GNN-GUIDED reduces to 2.15 cuts, closing approximately 55% of the gap. GNN-RAW reports 1.74 mean cuts, but only on the 106 structurally easier graphs it manages to solve, so this figure is not comparable to the others. Restricting all strategies to those same 106 graphs (Table 2b) removes the bias: the optimum falls to 1.09, and GNN-GUIDED (1.60 cuts) produces *fewer* cuts than GNN-RAW (1.74), closing 73% of the gap to the optimum versus 66% for the raw predictions. The repair procedure therefore improves partition quality on top of restoring feasibility, rather than degrading it; the apparent advantage of raw predictions on the full set was an artifact of their restriction to easier graphs.

Runtime. GNN-RAW is the fastest non-trivial strategy: partitioning takes 0.4 ms and SAT mapping 0.07 s, matching greedy's per-partition cost since both submit exactly one SAT check per partition. GNN-GUIDED (partitioning 4 ms, SAT mapping 0.24 s) is approximately 50× faster end-to-end than SAT-MIN-K (Algorithm 1) (12.1 s total), while the topological strategy (0.50 s SAT mapping) is slower than GNN-GUIDED despite producing worse partitions, because a poor seed drives more feasibility-repair iterations and consequently more SAT checks per partition (3.71 vs. 2.36).

5 Conclusion

This work presented a GNN-guided partitioning framework for spatial CGRA mapping that learns optimal partition structure from exhaustive SAT-generated ground truth. The proposed flow addresses the case where the DFG does not fit within the CGRA. While most approaches focus on accelerating placement and routing, this work targets the pre-mapping phase.

The framework makes three contributions. First a configurable synthetic DFG generator, which produces training graphs by applying structure-preserving mutations to kernel templates, enabling large-scale dataset generation without relying on compiler-extracted benchmarks. Second an exhaustive SAT-MIN-K (Algorithm 1) search provides provably optimal partition assignments as ground-truth labels, ensuring the GNN learns from high-quality supervision. Finally, a topology-aware guided partitioner which translates per-node GNN predictions into feasible partitions through convexification and capacity repair.

Evaluation on a 4×4 homogeneous CGRA with five strategies shows that the proposed GNN-GUIDED flow is the strongest practical method. On the full

evaluation set it achieves a 98% feasibility rate and reduces mean edge cuts by 33% relative to the unguided topological baseline, closing 55% of the gap to the SAT-MIN-K (Algorithm 1) optimum at roughly 50× lower runtime. A like-for-like comparison on the 106 graphs that GNN-RAW also solves (Table 2b) clarifies the role of the repair procedure: once graph structure is controlled for, GNN-GUIDED produces fewer cuts than GNN-RAW (1.60 vs. 1.74, against an optimum of 1.09), closing 73% of the gap to the optimum versus 66% for the raw predictions. The apparent quality advantage of GNN-RAW on the full set was therefore an artifact of its restriction to structurally easier graphs, not evidence that raw seeds are superior; repair both restores feasibility (98% vs. 70.7%) and lowers the cut count.

The large gap between the unguided topological baseline (3.00 cuts on the common subset) and both GNN-seeded variants confirms that the learned predictions carry strong partition structure, and the residual gap to the optimum reflects prediction error rather than a limitation of the repair mechanism. GNN-RAW remains the cheapest option, since it runs one SAT check per partition, but is dominated on both feasibility and cut quality, positioning it as a low-cost fallback rather than the primary method.

Several directions remain open. The current evaluation targets a 4×4 homogeneous CGRA; scaling to larger grids and heterogeneous architectures, where PEs support different ISA subsets, will stress both the GNN's generalization and the repair procedure's routing assumptions. An alternative GNN formulation predicting cut probability per edge in addition to partition index per node could close the remaining quality gap more directly. Finally, the framework currently assumes single-cycle, one-shot execution per partition; integrating temporal or modulo scheduling would substantially broaden its applicability to real compilation workflows.

Acknowledgments. This work is financed by National Funds through the FCT – Fundação para a Ciência e a Tecnologia, I.P. (Portuguese Foundation for Science and Technology) within the project AIMaCoV, with reference 2023.14899.PEX (`https://doi.org/10.54499/2023.14899.PEX`).

References

1. Hamilton, W.L., Ying, R., Leskovec, J.: Inductive representation learning on large graphs. In: Advances in Neural Information Processing Systems. vol. 30 (2017)
2. Karypis, G., Kumar, V.: A fast and highly quality multilevel scheme for partitioning irregular graphs. SIAM Journal on Scientific Computing **20**(1), 359–392 (1998). https://doi.org/10.1137/S1064827595287997
3. Kojima, T., Ohwada, A., Amano, H.: Mapping-aware kernel partitioning method for cgras assisted by deep learning. IEEE Transactions on Parallel and Distributed Systems **33**(5), 1213–1230 (2022). https://doi.org/10.1109/TPDS.2021.3107746
4. Li, J., Cai, C., Zhao, Y., Yan, Y., Yin, W., Wang, L.: Graft: Gnn-based adaptive framework for efficient cgra mapping. In: 2023 International Conference on Field Programmable Technology (ICFPT). pp. 26–34 (2023). https://doi.org/10.1109/ICFPT59805.2023.00008

5. Li, Z., Wu, D., Wijerathne, D., Mitra, T.: Lisa: Graph neural network based portable mapping on spatial accelerators. In: 2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA). pp. 444–459 (2022). https://doi.org/10.1109/HPCA53966.2022.00040
6. Mei, B., Vernalde, S., Verkest, D., De Man, H., Lauwereins, R.: Dresc: a retargetable compiler for coarse-grained reconfigurable architectures. In: 2002 IEEE International Conference on Field-Programmable Technology, 2002. (FPT). Proceedings. pp. 166–173 (2002). https://doi.org/10.1109/FPT.2002.1188678
7. Miyasaka, Y., Fujita, M., Mishchenko, A., Wawrzynek, J.: SAT-Based Mapping of Data-Flow Graphs onto Coarse-Grained Reconfigurable Arrays, pp. 113–131 (07 2021). https://doi.org/10.1007/978-3-030-81641-4_6
8. Tirelli, C., Ferretti, L., Pozzi, L.: Sat-mapit: A sat-based modulo scheduling mapper for coarse grain reconfigurable architectures. In: 2023 Design, Automation & Test in Europe Conference & Exhibition (DATE). pp. 1–6. IEEE (2023)