# SHAREGATE: LEARNING TO MATCH FUNCTIONAL SUBGRAPH WITH SHARED LOGIC

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Functional subgraph matching identifies circuit regions that implement the same Boolean functions as query patterns, supporting datapath optimization, arithmetic verification, and hardware Trojan analysis. Logic synthesis creates shared intermediate expressions, so functional regions overlap and gates serve multiple operations. This shared logic interferes with learned recognition: neighborhood aggregation mixes operation cues at shared gates, leaving gate classes ambiguous. To address this, we propose ShareGate, a framework that recovers and certifies arithmetic cores in post-synthesis netlists with shared logic. Within ShareGate, a graph neural network proposes a candidate region for each gate from per-gate descriptors. Verification-driven refinement accepts a region only when its simulated response reproduces a fixed reference model on every pattern in the budget. A scheduler orders the checks. Thus, a class label changes only through a passing functional test, and the accepted region carries that test as a certificate. We formalize the task as functional subgraph matching with shared logic. Each gate is assigned one of five classes (adder, multiplier, subtractor, comparator, and control), so arithmetic regions are labeled together with their embedded comparison and control logic. Experiments on 21 netlists with embedded arithmetic cores from EPFL, ISCAS-85, and ITC-99 show that ShareGate recovers and certifies all three arithmetic cores in every case at all four injection levels, reaching certified results $6.0$ to $34.5$ times faster than the fastest baseline. As a diagnostic of the recognition stage, the gate-level field reaches $92.3\%$ accuracy at the deepest level, versus $76.8\%$ for the strongest baseline.

## 1 INTRODUCTION

Functional subgraph matching identifies regions of a larger circuit that implement the same Boolean functions as query patterns. Recovering these regions supports datapath optimization, arithmetic verification, and hardware Trojan analysis, where subsequent analysis depends on knowing which gates implement a target computation (Salmani et al., 2013; Fyrbiak et al., 2019; Kaufmann & Biere, 2021; Mahzoon et al., 2022; Zheng et al., 2025).

In practice, functional subgraph matching must operate on the netlists after logic synthesis (Sasao, 1993), an essential stage in Electronic Design Automation (EDA) workflow. Synthesis preserves the specified behavior, but it can substantially change the structure and boundaries of the operations described in the source design. More importantly, as shown in Figure 1, *logic sharing* makes this task challenging because a gate can contribute to multiple operations. When different operations use the same intermediate expression, synthesis tools may implement that expression once and share the resulting gates across operations (Brayton et al., 1990; Mishchenko et al., 2006a) to save area.

Many approaches have attempted to recover functional regions from synthesized netlists. Pattern matching approaches recognize circuit functions by finding correspondences between a query template and a region of the netlist (Tygar & Ellickson, 1985; Ohlrich et al., 1993; Tashjian & Davoodi, 2015; Meade et al., 2016). Synthesis fuzzes the boundary of a query pattern, so a method that reads the topology alone becomes limited.

Other approaches train a Graph Neural Network (GNN) to predict gate classes or region boundaries (Alrahis et al., 2022; Wang et al., 2023; Li et al., 2025b; Liu et al., 2025; Zheng et al., 2025; Chen et al., 2026).However, shared logic interferes with these predictors at the aggregation step. Specifically, a GNN assigns a functional role to a gate by aggregating the features of its neighbours, and at a shared gate the neighbourhood holds the cues of every operation that meets there, so the aggregated representation mixes them and the predicted boundary moves across region edges. A more expressive GNN may improve these predictions, but the predicted labels alone do not establish whether the recovered region implements the target function. This gap motivates complementing



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

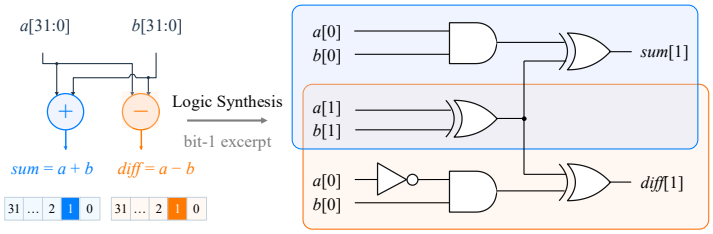


Figure 1: Logic sharing between addition and subtraction in an arithmetic and logic unit (ALU) (Bush, 2026). Both operations reuse the intermediate expression $a[1] \oplus b[1]$, implemented by a shared XOR gate. The blue and orange regions contain the gates computing $sum[1]$ and $diff[1]$, respectively.

learned recognition with functional checks: the model proposes candidate regions, while their simulated behavior provides evidence for accepting or revising the predictions.

In this work, we propose *ShareGate*, a framework for functional subgraph matching in circuits with shared logic. ShareGate combines learned recognition with verification-driven refinement (VDR), so that functional evidence directs the correction of regions whose cues aggregation mixes. In our proposed pipeline, a GNN first predicts, for every gate, one of five functional classes: adders, multipliers, subtractors, comparators, and control logic. The candidate regions are constructed from these predictions, and VDR then compares the simulated behavior of candidate regions with fixed reference models. A passing window determines the support cone whose labels are updated, while unsuccessful candidates are expanded for further checking or discarded. Finally, ShareGate combines accepted updates with the initial predictions outside accepted regions to produce a complete gate-level assignment. A label revision therefore occurs only through an explicit functional check, and the learned stage supplies the candidates that keep the search bounded. We evaluate ShareGate on 21 netlists from EPFL (Amaru et al., 2019), ISCAS-85 (Brglez & Fujiwara, 1985), and ITC-99 (Davidson, 1999) across four controlled levels of the sharegate injection. ShareGate recovers and certifies all three arithmetic cores in every evaluated case, and reaches the certified result $6.0$ to $34.5$ times faster than the fastest baseline. Its gate-level field reaches 92.3% node accuracy at the deepest level of the injection, against 76.8% for the strongest baseline. Section 4 separates the two measures and Appendix C documents the implementation constants.

Our contributions are summarized as follows:

- We formulate functional subgraph matching with shared logic as a gate-level annotation task, and we separate the ownership of a gate from the supports that reach its fan-in neighbourhood.
- We propose ShareGate, which pairs learned candidate recognition with verification-driven refinement. A simulation conformance check against a fixed reference model governs every label revision, and a UCB1 scheduler allocates a bounded checking budget.
- We evaluate arithmetic-core recovery and certification on public benchmarks under controlled share logic coupling. ShareGate certifies all three arithmetic cores in every case, and it reaches the certified result faster than nine published baselines under one fixed protocol.

## 2 Preliminaries

### 2.1 Circuit Netlists

A gate-level netlist is a directed acyclic graph (DAG) $G = (V, E)$, where each node $v \in V$ denotes a gate and an edge $(u, v) \in E$ indicates that the output of $u$ drives an input of $v$. A *functional region* is a set of gates that collectively implements a function with respect to specified input and output boundaries. For a combinational region $R \subseteq V$, these boundaries define a Boolean mapping $f_R : \{0,1\}^{m_R} \to \{0,1\}^{n_R}$, where $m_R$ and $n_R$ count the boundary inputs and outputs. The function of a region is distinct from the primitive operation of an individual gate: an XOR gate, for example, may contribute to both an addition and a subtraction.

Logic synthesis preserves the specified circuit behavior and can reuse an intermediate computation across operations to reduce power, performance and area (PPA) (Mishchenko et al., 2006a; Cocke, 1970; Brayton et al., 1990; Kubica et al., 2020). Let $S_i \subseteq V$ denote the support cone of an operation $i$ within its specified boundaries. *Logic sharing* occurs when two operations have intersecting supports, as Definition 1 states, and a shared gate then contributes to both computations while appearing once in the netlist. Recovering functional regions requires more than reading the connectivity.

**Definition 1** (Logic Sharing). *Under the designated supports $\{S_k\}$, $\kappa(v) = |\{k : v \in S_k\}|$ is the number of designated supports that contain the gate, and the overlapped set is $\mathcal{O} = \{v : \kappa(v) \geq 2\}$. A gate $v$ is* overlapped *if $v \in \mathcal{O}$, and two outputs* intersect *if they share a gate. Sharing also reaches gates beyond the supports that own them. Write $\mathcal{N}(v)$ for the fan-in neighbours of $v$, and let $\alpha(v) = |\{k : \mathcal{N}(v) \cap S_k \neq \varnothing\}|$ be the number of supports that reach the fan-in neighbourhood of $v$.*

$\kappa$ counts ownership, and a gate with $\kappa(v) \geq 2$ leaves its reference class open because two supports claim it. $\alpha$ counts what a predictor that aggregates over the fan-in neighbourhood reads, and a gate with $\alpha(v) \geq 2$ carries the cues of two operations in its input even when one support alone owns it. One synthesis step produces both, because implementing a shared expression once puts its gates into several supports and puts several supports into the neighbourhoods that read them.

$\kappa$ and $\alpha$ describe the circuit. What a predictor reads is not decided by the circuit alone, so we also write $\tilde{\mathcal{N}}(v)$ for the neighbours that the aggregation layer of the classifier actually reads, which in this implementation is an undirected neighbourhood, and

$$\tilde{\alpha}(v) = |\{k : \tilde{\mathcal{N}}(v) \cap S_k \neq \varnothing\}|. \tag{1}$$

An implementation that reads a symmetric neighbourhood reaches the gates a gate drives as well as the gates that drive it, so $\tilde{\alpha}$ is not $\alpha$ whenever a neighbour carries a support that no fan-in neighbour carries.

### 2.2 Functional Subgraph Matching

Circuit reverse engineering recovers functional blocks from gate-level implementations to support downstream analysis. RevSCA, for example, identifies half and full adders and uses their algebraic relations to accelerate multiplier verification (Mahzoon et al., 2022), and reduces verification time from $2{,}105.74$ s to $5.09$ s on a $64 \times 64$ Dadda-tree multiplier. Identifying functional regions also provides units for datapath optimization and hardware security analysis (Fyrbiak et al., 2019; Kaufmann & Biere, 2021; Salmani et al., 2013).

Functional subgraph matching seeks a region in a post-synthesis netlist that implements a target function or pattern (Zheng et al., 2025). Structural template methods correspond the nodes of a query pattern to those of the host graph, and they localize the pattern when its implementation structure survives synthesis (Ohlrich et al., 1993; Tygar & Ellickson, 1985; Meade et al., 2016; Tashjian & Davoodi, 2015). Under logic sharing the same target computation may take a different realization, and shared intermediate logic may participate in several candidate patterns. Learning-based gate classification predicts a functional category for each node from gate attributes and graph context, which annotates a region from the gate context alone (Alrahis et al., 2022; Li et al., 2025b; Liu et al., 2025). Its output is a field of node labels, and one category on a shared gate leaves open which regional computation supports that assignment. Arithmetic block identification describes a block by its inputs and outputs, and a boundary states the class of a gate in intersecting supports only through its supports (Wang et al., 2023). Both routes therefore motivate the distinction between recovering overlapping functional supports and assigning gate-level labels under logic sharing.

### 2.3 Problem Statement

We formulate functional subgraph matching under logic sharing as gate-level annotation. Functional supports may overlap, and the required output assigns one functional class to each gate. Assigning a shared gate to one class leaves its contribution to the other operations in place and leaves the disjointness of the underlying regions to the supports. Gates of one class may also belong to different instances of an operation, so a class label alone identifies a class and leaves the region open. Definition 2 fixes the complete assignment over the netlist, and evaluation compares predictions with the reference labels across all five classes, comparator and control included. The problem is therefore to recover a complete gate-level assignment under the ambiguity that shared functional supports introduce. The arithmetic cores are the objects a conformance checker decides, so the primary measures of this paper are the certified cores and the gate-level field serves as a diagnostic of the recognition stage. Appendix K.1 fixes the annotated gate set that every measured metric is scored on.

**Definition 2** (Gate-Level Annotation under Logic Sharing). *Given a post-synthesis netlist $G = (V, E)$, the task is to infer a complete label assignment $\hat{y} : V \to \mathcal{C}$, where $\mathcal{C} = \{\text{adder}, \text{multiplier}, \text{subtractor}, \text{comparator}, \text{control}\}$. Each gate $v \in V$ receives exactly one class, including shared gates that belong to the functional supports of multiple operations.*

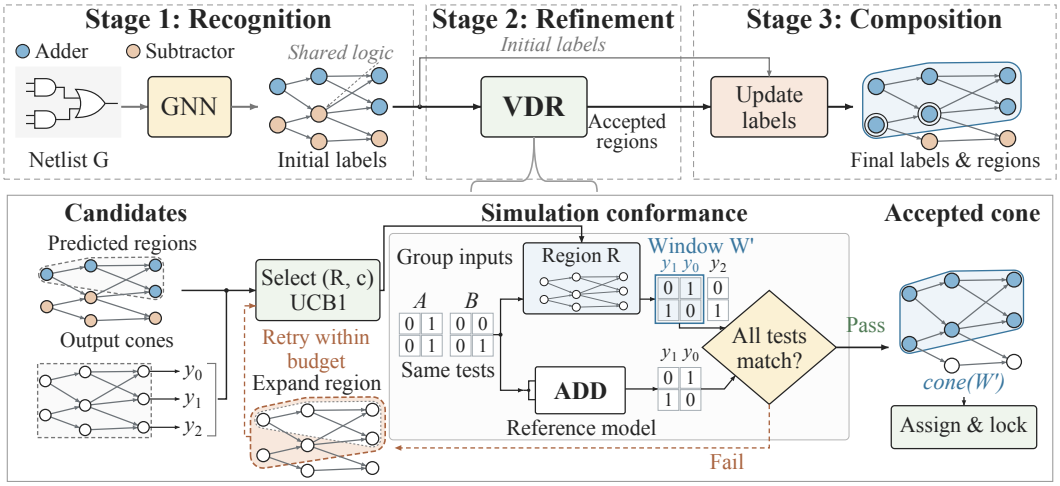


Figure 2: Overview of ShareGate. Recognition predicts gate classes, refinement checks a region-class hypothesis against a fixed reference model, and composition keeps the predictions outside an accepted cone. UCB1 orders the candidates. Hollow nodes leave the class blank, double rings mark a change, and nested outlines mark an expansion.

# 3 METHOD

## 3.1 OVERVIEW OF SHAREGATE

We propose *ShareGate*, a framework for functional subgraph matching in gate-level circuits with shared logic. Figure 2 shows its three stages. *Recognition* (§3.2) maps fixed per-gate feature vectors to a class field. *Refinement* (§3.3) is the contribution of this paper. It treats the field as a set of candidate regions, and a bounded simulation-conformance budget decides the regions it accepts. *Composition* (§3.4) merges the verified regions back into the field for downstream equivalence checking. Section 3.5 delimits the resulting guarantee. Appendix C gives the implementation constants and the algorithm.

## 3.2 RECOGNITION: CANDIDATE PROPOSAL

Recognition maps each gate to an initial class field $\hat{y}^{(0)} \in \mathcal{C}^{|V|}$ from a fixed per-gate descriptor of structural, topological and simulation-derived statistics. The stage is a GraphSAINT backbone (Zeng et al., 2020) fixed before evaluation and used unchanged in every experiment, and the candidate set it produces is the input of the refinement stage. Its descriptor and training protocol are in Appendix C.

## 3.3 REFINEMENT: VERIFICATION-DRIVEN CLASS REVISION

**Definition 3** (Conformance-checked window)**.** *For a set $W$ of output nets (a* window*), the* support cone $\mathrm{cone}(W) \subseteq V$ *is the transitive fan-in of $W$ restricted to $V$ (primary inputs excluded), with* $\mathrm{cone}(\varnothing) = \varnothing$*. Write* $\mathrm{resp}(W, x) \in \{0,1\}^W$ *for the values the nets of $W$ take under pattern $x$. Let $X_K$ be $K$ patterns drawn uniformly and independently from the primary inputs that feed* $\mathrm{cone}(W)$*, let $M_c$ be a class-$c$ reference model with a $\beta$-bit output bus, and let $|W| \le \beta$, so that an injective $g : W \to \{1, \dots, \beta\}$ puts every net of $W$ on its own bus position, with $M_c^g(x) \in \{0,1\}^W$ carrying at $v \in W$ the $g(v)$-th bus bit of $M_c$ under $x$. The window is* conformance-checked *for class $c$ under $g$ if* $\mathrm{resp}(W, x_k) = M_c^g(x_k)$ *for every $x_k \in X_K$ (Eq.* (3)*). The checker derives the grouping from the fixed hypothesis space, so a cone whose inputs also feed another region is accepted only under a grouping of the fixed hypothesis space that reproduces the model's response.*

**Candidate generation.** Two sources are united per class: support cones of multi-bit output buses, since a width-1 tap is an observation point, and same-class components of the prediction split by cosine-radius density clustering of the learned embedding.

**The verification procedure.** A UCB1 bandit (Auer et al., 2002) orders the checks under hard per-round and total budget caps (Appendix C): at round $t$ the scheduler plays the candidate with the

highest index

$$a_t = \arg\max_{a \in \mathcal{A}_t} \Big[ \hat{\mu}_a + \sqrt{2 \ln t \,/\, n_a} \Big], \tag{2}$$

where $\mathcal{A}_t \subseteq \mathcal{A}$ holds the candidates with fewer than three attempts against the cap, $n_a$ counts the attempts already made on candidate $a$, $\hat{\mu}_a$ is the fraction that passed, and an untried candidate takes priority while the check budget allows. The index orders the tests. Observation 1 in Appendix B bounds the number of selections.

Write $X_K = \{x_1, \ldots, x_K\}$ for the pattern budget of a round, drawn from the primary inputs feeding the region under test, and

$$\mathrm{viol}^{g}_{M_c}(W, X_K) = \sum_{k=1}^{K} \mathbf{1}\big[\mathrm{resp}(W, x_k) \neq M^g_c(x_k)\big] \tag{3}$$

for the number of patterns , both indexed by the nets of $W$ under $g$, the comparison is bit for bit and $\mathrm{viol}^{g}_{M_c}(W, X_K) = 0$ is the accept condition of Definition 3. For a candidate region $R$ with proposed class $c$, the checker simulates $R$ bit-parallel under $X_K$ against the reference models and searches output-window groupings for such a match. Formally, one round applies

$$\hat{y}^{(t+1)}(v) = \begin{cases} c^\star(R), & v \in \mathrm{cone}(W^\star), \\ \hat{y}^{(t)}(v), & \text{otherwise}, \end{cases}$$
$$(W^\star, g^\star) = \text{a maximum-cone pair of } \{(W, g) : W \in \mathcal{W}(R),\ \mathrm{viol}^{g}_{M_{c^\star}}(W, X_K) = 0\}, \tag{4}$$

where $\mathcal{W}(R)$ is the set of non-empty output windows of $R$ and $c^\star(R)$ is the class hypothesis the candidate carries. The pair taken is a passing pair of maximum $|\mathrm{cone}(W)|$, ties follow the search order fixed before evaluation, and $W^\star = \varnothing$ when every pair fails. An accepted window writes the verified class to $\mathrm{cone}(W^\star) \setminus L$, where $L$ holds the gates an accepted window has already written, and $L$ then becomes $L \cup \mathrm{cone}(W^\star)$. The locked set therefore grows monotonically, so every earlier assignment is overwritten. A failing window is retried under the next enumerated hypothesis and enlarged from seeds inside it, and a region leaves the candidate list once its attempt cap is reached. Refinement therefore returns a region only through a passing check, and Observation 1 bounds the number of selections it makes.

### 3.4 Composition

Accepted cone updates merge with the classifier field: gates inside a passing window take the verified class and every other gate keeps the prior, so the composed field is complete by construction. Each accepted update feeds back into the candidate list, so later rounds revisit an accepted cone and leave the remaining gates untouched, which keeps the check count flat as the injection deepens (§4.6).

**Proposition 1** (Locking of accepted gates)**.** *An accepted window writes only to* $\mathrm{cone}(W^\star_t) \setminus L$ *and extends* $L$ *by* $\mathrm{cone}(W^\star_t)$*, and composition leaves every gate outside an accepted cone in place, so a gate whose class an accepted window wrote at round* $t$ *carries that class at every later round:* $\hat{y}^{(\tau)}(v) = \hat{y}^{(t+1)}(v)$ *for every* $v \in \mathrm{cone}(W^\star_t)$ *and every* $\tau > t$*.*

**Proposition 2** (Conditional acceptance of an update)**.** *Every class change is produced by the checker and is backed by a zero-violation simulation conformance of a window of the candidate region against a reference model of the new class, so the revised gates are the gates of that cone outside the locked set, and acceptance is conditional on that budget and on the reference models.*

**Proposition 3** (Independence of the certified region from the labels of shared gates)**.** *Let* $R$ *be a candidate region with proposed class* $c$*, and let* $W$ *be a window of* $R$*. The accept condition of Definition 3 compares* $\mathrm{resp}(W, x_k)$ *with* $M^g_c(x_k)$ *for every* $x_k \in X_K$*, and both vectors are indexed by the nets of* $W$ *under* $g$*. A class label carried by a gate outside* $W$ *therefore leaves the violation count unchanged. The gates an accepted round writes are the gates of* $\mathrm{cone}(W^\star)$ *outside the locked set, so the certified region is fixed by the response of the window and by the reference model.*

*Proof.* The violation count of Eq. (3) sums the indicators of $\mathrm{resp}(W, x_k) \neq M^g_c(x_k)$ over the pattern budget, and both sides of that comparison are vectors over $W$. A label assignment on $V \setminus W$ leaves both vectors unchanged, and the count follows. Acceptance is the condition $\mathrm{viol}^{g}_{M_c}(W, X_K) = 0$, and the write of Eq. (4) targets $\mathrm{cone}(W^\star)$. The accepted region therefore follows from the pair $(W^\star, g^\star)$ and the reference model. □

### 3.5 Boundaries and Their Scope

Two checks bound methods of this kind. A *conformance check* simulates a region against a reference model on a finite budget, so zero violations over $K$ patterns is evidence relative to that budget, where a proof over every input would need a lower bound on the disagreement rate (Appendix B.3). *Equivalence checking* proves a property of the whole input space once a specification is supplied (Berman & Trevillyan, 1989; Kuehlmann & Krohm, 1997; Mishchenko et al., 2006b; Brayton & Mishchenko, 2010; Biere et al., 2021; Konrad & Scholl, 2024; Li et al., 2025a; Rigi-Luperti et al., 2026; Yang et al., 2026; Zhang et al., 2026; Hofstadler et al., 2026). Neither check recovers that specification, so this paper uses the first to decide a label, the second to certify a boundary and Proposition 2 to bound the reach of a refinement step that accepts. A run accepts a region only when its simulated response matches the reference model on every pattern of the budget, so the class a gate carries after a revision is the one an accepted window attests and a certified class comes from the checker, so classifier error alone leaves a class uncertified. Gates outside every accepted window keep the prior, so the check certifies the revision and the rest of the field is scored as the classifier's prediction. The reference functions are fixed before evaluation, and the designed miters are decided by SAT on the emitted AIGs (Appendix H). A region is accepted on a pattern coincidence whose probability Appendix B bounds in terms of the disagreement rate of the compared functions (Appendix C). Proposition 3 makes certification the measure a benchmark with shared logic can carry, and Propositions 1 and 2 make every revision a passing check.

## 4 Experiments

We conduct experiments to investigate the effectiveness of ShareGate in functional subgraph matching with shared logic. Section 4.1 states the settings and the measures, and Section 4.2 describes the data preparation. Section 4.5 measures the recovery and the certification of the arithmetic cores, which are the primary measures of this paper. Section 4.6 measures the time to a certified result. Section 4.3 measures the gate-level field as a diagnostic of the recognition stage, and Section 4.4 isolates the contribution of verification-driven refinement.

### 4.1 Experiment Setup

**Datasets** We train on 382 gate-level netlists containing approximately 193,000 annotated gates. For evaluation, we use 21 open-source designs with 4,086–7,634 gates from EPFL (Amaru et al., 2019), ISCAS-85 (Brglez & Fujiwara, 1985), and ITC-99 (Davidson, 1999). Each design contains three freshly synthesized arithmetic cores, a 40-bit adder, a 40-bit subtractor, and a 20-bit multiplier, together with a 40-bit comparator and the control background, so all five functional classes occur in every evaluation netlist. Appendix E lists the netlists.

**Baselines and metrics** We compare ShareGate with 9 prior methods: ABGNN (Wang et al., 2023), DAGNN-RE (Li et al., 2025b), GAT (Veličković et al., 2018), GCN (Kipf & Welling, 2017), GraphSAGE (Hamilton et al., 2017), HGCN (Chami et al., 2019), NeuroMatch (Lou et al., 2020), ReVEAL (Chen et al., 2026), and WideGate (Liu et al., 2025). A majority predictor assigns the most frequent class of the training pool to every gate, and it fixes the height a method reaches from class frequency alone. Performance is evaluated using node classification accuracy (Acc), macro-F1 (mF1), multiclass Matthews correlation coefficient (MCC), and balanced accuracy (bAcc). Macro-F1 averages the F1 scores of the five classes with equal weight per class. All metrics are computed on gates with defined reference labels.

**Evaluation Analysis** The evaluated netlists annotate every gate, so class metrics in Section 4.3 cover the entire post-synthesis netlist. Section 4.2 defines the controllable operation that injects overlap at every level of $\rho$. Logic sharing affects both the gates that read a region and the gates the region owns, threatening the recovered arithmetic core. Since a class label alone cannot determine a region's function, a conformance check is required. Thus, the primary measures are the recovery and certification rates of arithmetic cores, while gate-level metrics diagnose recognition. Proposition 3 states the property that makes the certified region the measure of what logic sharing leaves intact. The *recovery rate* counts how many of the three arithmetic cores receive a region, certified or pending; it is written as $k/3$ to keep partial results visible. The *certification rate* counts recovered cores whose regions pass the equivalence check in Appendix K.6; a case is solved only when all three cores are certified. The *time to target* is wall-clock time from case start to certification of the third core. All arms use one fixed protocol, one fixed cap, and one fixed checker.

**Implementation** Training runs in full batch mode with Adam at a learning rate of 0.01 on two NVIDIA A100 GPUs. The other implementation details can be found at Appendix C.

Table 1: Results of gate-level classification

| | $\rho$=0 | | | | $\rho$=0.25 | | | | $\rho$=0.50 | | | | $\rho$=0.75 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc | mF1 | MCC | bAcc | Acc | mF1 | MCC | bAcc | Acc | mF1 | MCC | bAcc | Acc | mF1 | MCC | bAcc |
| Majority | 75.25 | 17.12 | 0.00 | 20.00 | 75.25 | 17.12 | 0.00 | 20.00 | 75.25 | 17.12 | 0.00 | 20.00 | 75.25 | 17.12 | 0.00 | 20.00 |
| ABGNN | 95.89 | 89.38 | 90.85 | 92.03 | 86.47 | 55.62 | 63.91 | 56.56 | 79.98 | 30.61 | 42.14 | 35.28 | 75.70 | 24.94 | 33.27 | 31.65 |
| DAGNN-RE | 94.52 | 88.22 | 87.67 | 93.30 | 87.92 | 61.61 | 71.95 | 73.08 | 82.97 | 51.16 | 62.25 | 66.33 | 76.80 | 45.61 | 52.29 | 61.31 |
| GAT | 91.53 | 84.72 | 81.24 | 92.05 | 80.53 | 50.94 | 55.46 | 55.07 | 72.80 | 31.95 | 39.28 | 36.86 | 65.15 | 23.46 | 28.77 | 30.53 |
| GCN | 82.76 | 77.48 | 66.24 | 87.20 | 57.17 | 42.26 | 33.81 | 54.06 | 50.22 | 30.48 | 24.80 | 41.33 | 45.71 | 24.74 | 20.11 | 34.71 |
| GraphSAGE | 92.85 | 86.80 | 84.41 | 94.06 | 75.79 | 51.58 | 52.76 | 61.98 | 54.78 | 33.47 | 32.78 | 45.07 | 33.31 | 22.49 | 19.47 | 33.61 |
| HGCN | 77.43 | 67.10 | 58.67 | 79.35 | 73.62 | 45.27 | 43.65 | 50.06 | 70.38 | 33.91 | 33.07 | 37.25 | 69.52 | 28.50 | 27.62 | 31.86 |
| NeuroMatch | 76.49 | 51.12 | 50.49 | 59.83 | 75.09 | 40.03 | 44.03 | 47.61 | 69.76 | 32.57 | 35.70 | 42.37 | 63.54 | 28.62 | 29.68 | 39.86 |
| ReVEAL | 54.82 | 38.61 | 22.42 | 43.06 | 49.53 | 27.01 | 15.82 | 31.52 | 50.84 | 23.94 | 13.50 | 27.62 | 55.16 | 23.72 | 13.09 | 26.95 |
| WideGate | 91.40 | 83.35 | 80.64 | 90.55 | 83.20 | 54.67 | 58.61 | 57.95 | 77.77 | 35.65 | 41.62 | 38.51 | 74.64 | 26.61 | 33.29 | 32.24 |
| **Ours** | **97.02** | **92.86** | **93.83** | **94.52** | **93.47** | **76.36** | **84.24** | **80.84** | **92.33** | **68.91** | **81.06** | **76.42** | **92.34** | **67.88** | **80.82** | **75.64** |

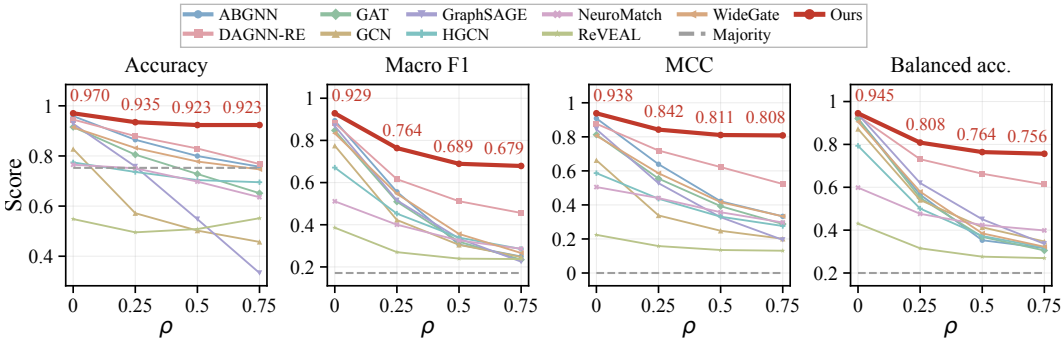


Figure 3: Performance of gate-level classification in different levels of $\rho$

### 4.2 Data Preparation

To evaluate recognition under shared logic, each design is flattened and synthesized with Yosys Wolf et al. (2013). Shared logic makes a gate's neighborhood carry evidence from multiple operations, so a neighborhood aggregator cannot infer a gate's class from its surroundings. To control this displacement, we append two-input observation gates that belong to no support. Each gate adds a neighbor carrying no class evidence, diluting the same neighborhood from the other side. Synthesis-induced sharing remains untouched, so the two effects are separated by construction. Each appended gate takes the outputs of two uniformly sampled gates and produces a fresh observation output. The injection changes nothing else; removing the appended gates restores the netlist gate for gate. Definition 4 fixes the number of appended gates, and we evaluate $\rho \in \{0, 0.25, 0.5, 0.75\}$.

**Definition 4** (Node Injection)**.** *For a post-synthesis netlist with $n$ gates before injection, $\rho$ fixes the number of appended observation gates as $q = \mathrm{round}(\rho n)$. Each appended gate has two inputs, the outputs of two uniformly sampled gates, and one output, a fresh observation output.*

Two properties determine what the sweep varies and holds fixed. The injection leaves every support assignment in place, so $\mathcal{O}$ and $\kappa$ are identical at every level of $\rho$: an appended XOR gate belongs to no support and carries $\kappa = 0$. The two reach quantities separate. $\alpha$ is also identical at every level of $\rho$, because an appended gate drives no gate and enters no fan-in neighborhood. $\tilde{\alpha}$ is not, because the aggregation layer reads a symmetric neighborhood in which an appended gate neighbors the two gates it taps. The sweep therefore varies $\tilde{\alpha}$ and the share of that neighborhood carrying no support.

Every gate in the post-synthesis netlist carries one reference class. The five class counts in Table 5 sum to the gate count of each evaluated netlist, so the scored set is the whole netlist before injection and is the same at every level of $\rho$. An appended observation gate lies outside the scored set because it lies outside every support and carries $\kappa = 0$ under Definition 1. Appendix K.1 states the annotation protocol that fixes the scored set.

### 4.3 Results of Gate-Level Classification

This subsection measures ShareGate's assignment of the five functional classes in the presence of shared logic. As shown in Table 1, ShareGate is compared with the other baselines and with a majority-class predictor at every $\rho$. The metrics with underlines represent the best results achieved by one of the 9 baseline methods. The majority predictor assigns one class to each gate, so its MCC is 0.00 at every level.

Three observations follow from Table 1 and Figure 3. First, ShareGate achieves the best results on all four metrics at every evaluated overlap level, which supports its effectiveness on post-synthesis

Table 2: Ablation ladder for Ours. The classifier our features support and the two class-space smoothing arms (Voting, MRF) are the alternative arms; the last row applies verification-driven refinement on the same prior.

| Method | $\rho = 0$ | | | | $\rho = 0.25$ | | | | $\rho = 0.5$ | | | | $\rho = 0.75$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc | mF1 | MCC | bAcc | Acc | mF1 | MCC | bAcc | Acc | mF1 | MCC | bAcc | Acc | mF1 | MCC | bAcc |
| Classifier | 83.84 | 81.26 | 69.37 | 90.85 | 83.20 | 67.16 | 64.25 | 74.66 | 78.61 | 53.36 | 54.30 | 59.44 | 74.38 | 42.62 | 46.40 | 46.99 |
| Voting | 86.22 | 83.53 | 72.89 | 91.98 | 85.83 | 69.39 | 68.49 | 76.05 | 80.87 | 54.35 | 57.03 | 58.91 | 76.41 | 41.01 | 47.78 | 44.08 |
| MRF | 85.34 | 83.15 | 71.64 | 91.74 | 84.88 | 68.57 | 66.99 | 75.71 | 80.55 | 54.46 | 56.90 | 60.01 | 76.41 | 42.71 | 48.53 | 46.45 |
| **Ours** | **97.02** | **92.86** | **93.83** | **94.52** | **93.47** | **76.36** | **84.24** | **80.84** | **92.33** | **68.91** | **81.06** | **76.42** | **92.34** | **67.88** | **80.82** | **75.64** |

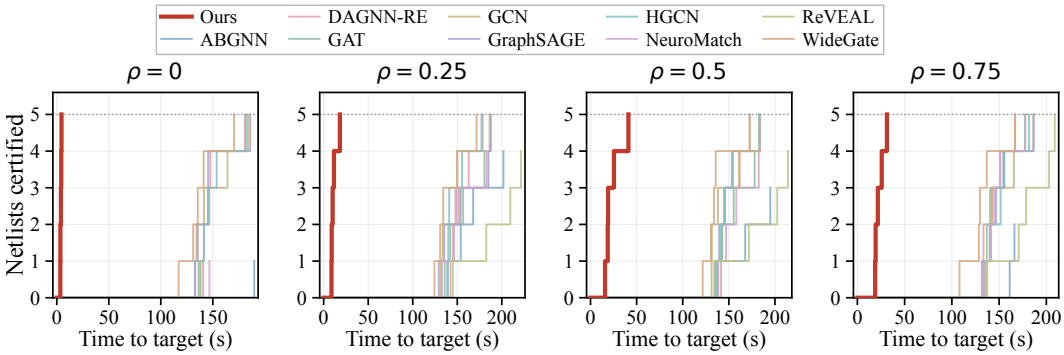


Figure 4: Certified netlists against time to target, one panel per level of $\rho$ , over the five pre-declared netlists . A netlist counts as certified when all three of its arithmetic cores certified.

netlists with shared logic. For instance, the strongest baseline achieves $76.80\%$ Acc and $45.61\%$ mF1, while ShareGate increases the accuracy to $92.34\%$ and macro-F1 to $67.88\%$, at the deepest level of the injection. Second, ShareGate also performs best at $\rho = 0$, where the netlist carries only the sharing that synthesis introduced. Third, Figure 3 shows the sensitivity of classification performance to increasing sharegate injection. Each panel plots one metric against $\rho$, comparing our proposed ShareGate with the baselines. Although ShareGate's performance also decreases as $\rho$ increases, it remains ahead across all four metrics, with a larger margin over the strongest baselines at $\rho = 0.75$. Therefore, ShareGate is more robust to increasing logic sharing than the prior methods.

### 4.4 Ablation Study

We further analyze the impact of verification-driven refinement by comparing the complete ShareGate system with three variants that share one prior field. *Classifier* directly outputs the initial gate labels from the prior field alone. *Voting* replaces refinement with graph-based label voting, while *MRF* applies Markov random field smoothing to the predicted classes. Both alternatives operate on the class predictions alone.

As shown in Table 2, first, a graph operator over the class space holds the field close to its prior. At $\rho = 0$, the prior reaches MCC 69.37, the two smoothing settings reach 72.89 and 71.64, and ours reaches 93.83. Each smoothing arm remains within roughly four points of its prior at $\rho = 0$ and two points at $\rho = 0.75$, so the graph operator moves the field by a small amount at every level. Second, this separation widens with overlap: refinement exceeds the best smoothing arm by +20.9 MCC at $\rho = 0$ and by +32.3 at $\rho = 0.75$, while the spread between smoothing settings is only 0.75 and 1.25 points. The three arms share one prior, so the difference in Table 2 belongs to the decision rule.

Smoothing sharpens a proposal, and the classes it can produce are the classes the prior proposes, while refinement admits a class only on zero violations and can therefore replace a proposal with a verified class. Table 2 attributes the gap to refinement under one prior and one graph budget, and Proposition 2 states the property that the smoothing arms lack.

### 4.5 Results of Functional Boundary Recovery

This subsection measures recovery and certification of arithmetic cores. A label field assigns each gate a category. A region becomes a boundary only if its gates form a closed cut whose module can be emitted and compared against a reference core. We recover the three arithmetic cores of each netlist and verify every region by equivalence checking against the reference core of the clean netlist.

The injections of Section 4.2 leave the recovered region a superset of the reference core. The recovered side is a gate-level implementation from the deformed netlist; the reference side instantiates the RTL core. Both use the same emitter over the reference core's port list, so the two and-inverter graphs share one declared interface rather than two independently derived ones. The verdict thus

Table 3: Each arm is equipped with a 300-second limit and a certification that is verified through traversal as a tool. *Certified of 5* counts the pre-declared netlists whose three arithmetic cores all certified, at each level of $\rho$. Times are medians over completed cases. The measures are defined in Section 4.1.

| Method | Certified of 5, per $\rho$ | Median completion time (s) | | | | Mean over $\rho$ |
|---|---|---|---|---|---|---|
| | | $\rho$=0 | $\rho$=0.25 | $\rho$=0.5 | $\rho$=0.75 | |
| ABGNN | 1,4,3,2 | 189.8 | 160.8 | 167.7 | 163.7 | 170.5 |
| DAGNN-RE | 5,5,5,5 | 146.6 | 151.8 | 145.0 | 142.9 | 146.6 |
| GAT | 1,4,4,5 | 138.3 | 149.3 | 148.9 | 140.5 | 144.3 |
| GCN | 5,5,5,5 | 141.5 | 148.1 | 138.2 | 142.0 | 142.4 |
| GraphSAGE | 5,5,5,5 | 145.2 | 153.9 | 145.8 | 146.7 | 147.9 |
| HGCN | 5,5,5,5 | 146.1 | 140.9 | 144.9 | 151.7 | 145.9 |
| NeuroMatch | 1,4,4,5 | 146.8 | 147.9 | 152.6 | 144.9 | 148.0 |
| ReVEAL | 1,4,4,5 | 137.1 | 196.0 | 187.1 | 178.6 | 174.7 |
| WideGate | 5,5,5,5 | 135.6 | 134.2 | 133.9 | 129.7 | 133.3 |
| **Ours** | **5,5,5,5** | **3.93** | **10.09** | **19.05** | **21.65** | **13.68** |

concerns the recovered region alone: it asks whether the region drives the reference core's boundary with the reference core's function. Appendix H provides the plain-miter lane that measures the solver's search. Across all four overlap levels, ShareGate certifies all three cores in every netlist.

Table 3 aggregates this result: ShareGate certifies all five pre-declared netlists at every overlap level and reaches that count fastest on all twenty cases. Appendix K.6 gives per-core accounting; Appendix H gives solver cost. The classifier outputs category labels, but the certificate requires verifiable functional boundaries. Certifying regions induced by the classifier's own field succeeds on zero cases because those regions form open cuts. Such regions may include extraneous functions or omit key ports, causing equivalence checks to fail. Refinement converts a per-node assignment into a solver-accepted boundary, turning the classification of Section 4.3 into a certified region. With the boundary supplied, the plain-miter lane of Appendix H is 84/84 UNSAT, so the solver keeps pace. Proposition 3 explains the separation: the certified region follows from the window response and the reference model, so labels the classifier carries on shared gates stay outside the check.

### 4.6 Efficiency Analysis

This subsection measures the cost of the primary result. Every arm runs under the same 300-second per-case limit and the same judging rule. The measured quantity is the time to a certified boundary.

Table 3 shows the result. ShareGate certifies all five panel netlists at every overlap level and is the fastest arm on all twenty cases, by $34.5\times$ at $\rho = 0$ and $6.0\times$ at $\rho = 0.75$. At $\rho = 0$, four baselines certify one of the five netlists. Figure 4 shows the same separation at the remaining levels: three baselines recover as overlap increases, while one cell exhausts the limit with zero certifications. The margin narrows as $\rho$ grows, because our cost rises with overlap.

Other baselines traverse the labels they predict to form a closed region, which takes hundreds of seconds per case. ShareGate instead proposes a boundary and tests it, so a bounded number of conformance checks replaces that traversal. Specific costs are broken down in Appendix H. The remaining time is the cost of the refinement stage, and the bound on the check count keeps that cost flat as interference grows (Proposition 1).

## 5 Conclusion

This paper formulates the functional subgraph matching task in post-synthesis netlists with shared logic. Shared logic interferes with learned recognition at the aggregation step and leaves the class of a shared gate open, so this paper measures the arithmetic cores that a conformance check can decide. We propose ShareGate, which pairs GNN-based candidate recognition with verification-driven refinement. A conformance check against a fixed reference model governs every label revision, and the predictions outside accepted regions complete the assignment. Experiments on 21 open-source netlists across four evaluation settings under shared logic show that ShareGate recovers and certifies all three arithmetic cores in every case, and reaches the certified result $6.0$ to $34.5$ times faster than the fastest baseline. The recognition stage supplies the field that refinement starts from, and it outperforms nine learned baselines on all four classification metrics, reaching $92.3\%$ node accuracy at the deepest level of injected sharing against $76.8\%$ for the strongest baseline. Beyond functional subgraph matching, these results suggest that ShareGate has the potential of improving EDA applications where the functional regions can be matched after logic synthesis.

## REFERENCES


Lilas Alrahis, Abhrajit Sengupta, Johann Knechtel, Satwik Patnaik, Hani Saleh, Baker Mohammad, Mahmoud Al-Qutayri, and Ozgur Sinanoglu. GNN-RE: Graph neural networks for reverse engineering of gate-level netlists. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(8):2435–2448, 2022. doi: 10.1109/TCAD.2021.3110807.

Luca Amaru, Pierre-Emmanuel Gaillardon, Eleonora Testa, and Giovanni De Micheli. The EPFL combinational benchmark suite. Zenodo, 2019. URL https://zenodo.org/records/2572934. Benchmark release associated with IWLS 2015.

Peter Auer, Nicolò Cesa-Bianchi, and Paul Fischer. Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, 47(2–3):235–256, 2002. doi: 10.1023/A:1013689704352.

C. Leonard Berman and Louise H. Trevillyan. Functional comparison of logic designs for vlsi circuits. In *IEEE International Conference on Computer-Aided Design (ICCAD)*, pp. 456–459, 1989. doi: 10.1109/ICCAD.1989.76990.

Armin Biere, Mathias Fleury, and Maximilian Heisinger. CaDiCaL, Kissat, Paracooba entering the SAT competition 2021. In *Proceedings of SAT Competition 2021: Solver and Benchmark Descriptions*, volume B-2021-1 of *Department of Computer Science Report Series B*, pp. 10–13. University of Helsinki, 2021. URL https://cca.informatik.uni-freiburg.de/papers/BiereFleuryHeisinger-SAT-Competition-2021-solvers.bib.

Robert Brayton and Alan Mishchenko. ABC: An academic industrial-strength verification tool. In *Computer Aided Verification (CAV), LNCS 6174*, pp. 24–40, 2010. doi: 10.1007/978-3-642-14295-6_5.

Robert K. Brayton, Gary D. Hachtel, and Alberto L. Sangiovanni-Vincentelli. Multilevel logic synthesis. *Proceedings of the IEEE*, 78(2):264–300, 1990. doi: 10.1109/5.52213.

F. Brglez and H. Fujiwara. A neutral netlist of 10 combinational benchmark circuits and a target translator in fortran. In *Proceedings of the IEEE International Symposium on Circuits and Systems (ISCAS 85)*, pp. 677–692. IEEE Press, 1985. URL https://ddd.fit.cvut.cz/www/prj/Benchmarks/ISCAS85.bib.

Jeff Bush. RISC-Processor: Arithmetic logic unit. https://github.com/jbush001/RISC-Processor, 2026. File verilog/rtl/alu.v, revision 9542d3a18f8c4224ef6865bb224e2cefe35a731f. Accessed September 19, 2026.

Ines Chami, Zhitao Ying, Christopher Ré, and Jure Leskovec. Hyperbolic graph convolutional neural networks. In *Advances in Neural Information Processing Systems 32 (NeurIPS)*, volume 32, pp. 4869–4880, 2019. URL https://arxiv.org/abs/1910.12933.

Chen Chen, Daniela Kaufmann, Chenhui Deng, Zhan Song, Hongce Zhang, and Cunxi Yu. ReVEAL: GNN-guided reverse engineering for formal verification of optimized multipliers. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2026), Lecture Notes in Computer Science*, pp. 309–333, 2026. doi: 10.1007/978-3-032-22749-2_16. URL https://arxiv.org/abs/2512.22260.

John Cocke. Global common subexpression elimination. In *Proceedings of a Symposium on Compiler Optimization*, pp. 20–24, 1970. doi: 10.1145/800028.808480.

Scott Davidson. Itc'99 benchmark circuits – preliminary results. In *International Test Conference 1999*, pp. 1125–1125, 1999. doi: 10.1109/TEST.1999.805857.

Marc Fyrbiak, Sebastian Wallat, Pawel Swierczynski, Max Hoffmann, Sebastian Hoppach, Matthias Wilhelm, Tobias Weidlich, Russell Tessier, and Christof Paar. HAL—the missing piece of the puzzle for hardware reverse engineering, trojan detection and insertion. *IEEE Transactions on Dependable and Secure Computing*, 16(3):498–510, 2019. doi: 10.1109/TDSC.2018.2812183.

William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 30, pp. 1024–1034, 2017. URL https://arxiv.org/abs/1706.02216.

Clemens Hofstadler, Daniela Kaufmann, and Chen Chen. Avoiding big integers: Parallel multimodular algebraic verification of arithmetic circuits. *arXiv preprint arXiv:2603.09501*, 2026.

Daniela Kaufmann and Armin Biere. AMulet 2.0 for verifying multiplier circuits. In *Proceedings of Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pp. 357–364, 2021. doi: 10.1007/978-3-030-72013-1_19.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *International Conference on Learning Representations (ICLR)*, 2017. URL https://arxiv.org/abs/1609.02907.

Alexander Konrad and Christoph Scholl. Symbolic computer algebra for multipliers revisited – it's all about orders and phases. In *2024 Formal Methods in Computer-Aided Design (FMCAD)*, pp. 261–271, Prague, Czech Republic, 2024. doi: 10.34727/2024/isbn.978-3-85448-065-5_32. URL https://abs.informatik.uni-freiburg.de/papers/2024/KS_2024.pdf.

Marcin Kubica, Adam Opara, and Dariusz Kania. Partial sharing of logic resources. In *Technology Mapping for LUT-Based FPGA*, Lecture Notes in Electrical Engineering, pp. 93–113. Springer, nov 2020. doi: 10.1007/978-3-030-60488-2_9. Published 8 November 2020.

Andreas Kuehlmann and Florian Krohm. Equivalence checking using cuts and heaps. In *Proceedings of the 34th Design Automation Conference (DAC)*, pp. 263–268, 1997. doi: 10.1145/266021.266090.

Rui Li, Lin Li, Heng Yu, Masahiro Fujita, Weixiong Jiang, and Yajun Ha. RefSCAT: Formal verification of logic-optimized multipliers via automated reference multiplier generation and SCA-SAT synergy. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 44 (2):791–804, 2025a. doi: 10.1109/TCAD.2024.3442987.

Zongtai Li, Liang Yang, and Mian Lou. DAGNN-RE: Directed acyclic graph neural network for functional reverse engineering of gate-level netlist. *Integration, the VLSI Journal*, 102:102343, 2025b. doi: 10.1016/j.vlsi.2024.102343.

Jiawei Liu, Zhiyan Liu, Xun He, Jianwang Zhai, Zhengyuan Shi, Qiang Xu, Bei Yu, and Chuan Shi. WideGate: Beyond directed acyclic graph learning in subcircuit boundary prediction. In *Design, Automation & Test in Europe Conference (DATE)*, pp. 1–7, 2025. doi: 10.23919/DATE64628.2025.10992972.

Zhaoyu Lou, Jiaxuan You, Chengtao Wen, Arquimedes Canedo, and Jure Leskovec. Neural subgraph matching. *arXiv preprint arXiv:2007.03092*, 2020.

Alireza Mahzoon, Daniel Große, and Rolf Drechsler. RevSCA-2.0: SCA-based formal verification of nontrivial multipliers using reverse engineering and local vanishing removal. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(5):1573–1586, 2022. doi: 10.1109/TCAD.2021.3083682.

Travis Meade, Shaojie Zhang, and Yier Jin. Netlist reverse engineering for high-level functionality reconstruction. In *21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 655–660, 2016. doi: 10.1109/ASPDAC.2016.7428086.

Alan Mishchenko, Satrajit Chatterjee, and Robert Brayton. DAG-aware AIG rewriting: A fresh look at combinational logic synthesis. In *Proceedings of the 43rd Annual Design Automation Conference*, pp. 532–535, 2006a. doi: 10.1145/1146909.1147048.

Alan Mishchenko, Satrajit Chatterjee, Robert Brayton, and Niklas Een. Improvements to combinational equivalence checking. In *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 836–843, 2006b. doi: 10.1109/ICCAD.2006.320087.

Miles Ohlrich, Carl Ebeling, Eka Ginting, and Lisa Sather. SubGemini: Identifying subcircuits using a fast subgraph isomorphism algorithm. In *Proceedings of the 30th ACM/IEEE Design Automation Conference (DAC)*, pp. 31–37, 1993. doi: 10.1145/157485.164556.

Niccolò Rigi-Luperti, Armin Biere, and Dominik Schreiber. Parallel SAT sweeping on CNFs. In *Proceedings of the 26th Conference on Formal Methods in Computer-Aided Design (FMCAD)*, pp. 182–188. TU Wien Academic Press, 2026. doi: 10.34727/2026/isbn.978-3-85448-093-8_24. URL https://repositum.tuwien.at/handle/20.500.12708/230486.

Hassan Salmani, Mohammad Tehranipoor, and Ramesh Karri. On design vulnerability analysis and trust benchmarks development. In *IEEE 31st International Conference on Computer Design (ICCD)*, pp. 471–474, 2013. doi: 10.1109/ICCD.2013.6657085.

Tsutomu Sasao (ed.). *Logic Synthesis and Optimization*. The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, Boston, MA, 1993. doi: 10.1007/978-1-4615-3197-0.

Edward Tashjian and Azadeh Davoodi. On using control signals for word-level identification in a gate-level netlist. In *Proceedings of the 52nd Annual Design Automation Conference (DAC)*, pp. 1–6, 2015. doi: 10.1145/2744769.2744878.

J. D. Tygar and Ron Ellickson. Efficient netlist comparison using hierarchy and randomization. In *Proceedings of the 22nd ACM/IEEE Design Automation Conference (DAC)*, pp. 702–708, 1985. doi: 10.1109/DAC.1985.1586019.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International Conference on Learning Representations (ICLR)*, 2018. URL https://arxiv.org/abs/1710.10903.

Ziyi Wang, Zhuolun He, Chen Bai, Haoyu Yang, and Bei Yu. Efficient arithmetic block identification with graph learning and network-flow. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(8):2591–2603, 2023. doi: 10.1109/TCAD.2022.3227815.

Clifford Wolf, Johann Glaser, and Johannes Kepler. Yosys-a free verilog synthesis suite. In *Proceedings of the 21st Austrian Workshop on Microelectronics (Austrochip)*, volume 97, pp. 1–6, 2013.

Ziyi Yang, Guangyu Hu, Xiaofeng Zhou, Mingkai Miao, Changyuan Yu, Wei Zhang, and Hongce Zhang. FORWORD: Accelerating formal datapath verification via word-level sweeping. In *Design, Automation & Test in Europe Conference (DATE)*, pp. 1–7, 2026. doi: 10.23919/DATE69613.2026.11539347.

Hanqing Zeng, Hongkuan Zhou, Ajitesh Srivastava, Rajgopal Kannan, and Viktor Prasanna. GraphSAINT: Graph sampling based inductive learning method. In *International Conference on Learning Representations (ICLR)*, 2020. URL https://arxiv.org/abs/1907.04931.

Xindi Zhang, Furong Ye, Zhihan Chen, and Shaowei Cai. FastLEC: Parallel datapath equivalence checking with hybrid engines. In *International Symposium on Formal Methods (FM), Lecture Notes in Computer Science*, pp. 89–108, 2026. doi: 10.1007/978-3-032-26204-2_5.

Ziyang Zheng, Kezhi Li, Zhengyuan Shi, and Qiang Xu. Functional matching of logic subgraphs: Beyond structural isomorphism. In *Advances in Neural Information Processing Systems 38*, pp. 110649–110671, 2025. doi: 10.52202/085713-3336. URL https://arxiv.org/abs/2505.21988.

### Reproducibility Statement

The artifact contains the netlist generators, the injection model, the functional class models, the conformance checker, and the training and evaluation code needed to reproduce every number given here. Appendix C documents the training protocol and the verification budgets.

### Use of Large Language Models

Large language models were used in three supporting roles: modifying scientific figures and tables, editing code, and polishing parts of the manuscript text. All scientific content, analysis and conclusions are the authors' own, and the authors reviewed and verified every modification made with these tools.

## A Symbols, Abbreviations and Their Roles

**Table 4:** Symbols and abbreviations, their expansions and their role in this task. Entries above the rule are the symbols used throughout the paper, with the place their meaning is fixed named in the last column. Entries below it are the abbreviations the paper uses.

| Symbol | Expansion | Role in this paper |
|---|---|---|
| $G = (V, E)$ | gate-level netlist DAG | $V$ gates, $E$ driver-to-sink edges |
| $\mathcal{C}$ | the five classes | {adder, multiplier, subtractor, comparator, control}, the formal task of Definition 2. Every measured metric is measured over the five classes of $\mathcal{C}$, so comparator and control gates are scored like the others. Comparator and control instances occur in the designs and their annotated counts are listed with the netlists |
| $y, \hat{y}$ | reference and predicted class fields | maps $V \to \mathcal{C}$. The output field is $\hat{y}$ over every gate that carries a reference class |
| $S_k \subseteq V$ | support of annotated core $k$ | the gate set a recovered region must reproduce |
| $C_j$ | same-class connected component | induced by $\hat{y}$. Candidates are unions of these |

Continued on next page

Table 4 (continued)

| Symbol | Expansion | Role in this paper |
|---|---|---|
| $\rho$ | sharegate injection | the number of appended observation gates, $q = \mathrm{round}(\rho n)$ for the gate count $n$ of the post-synthesis netlist before injection, and each appended gate has two sampled inputs and one fresh observation output (§4.2) |
| $W$ | window | set of output nets a check tests, with support cone $\mathrm{cone}(W)$ and $\mathrm{cone}(\varnothing) = \varnothing$ (§3.3) |
| $X_K$ | pattern set | $K$ patterns drawn uniformly and independently from the primary inputs that feed the cone under test |
| $\mathrm{resp}(W, x)$ | window response | a vector in $\{0,1\}^W$ of the values the nets of $W$ take under pattern $x$, determined by the pattern at the primary inputs feeding $\mathrm{cone}(W)$ and empty for $W = \varnothing$ (Definition 3) |
| $\kappa(v), \mathcal{O}$ | gate ownership | $\kappa(v) = \lvert\{k : v \in S_k\}\rvert$ and $\mathcal{O} = \{v : \kappa(v) \geq 2\}$, the gates that two functional supports claim (Definition 1) |
| $\alpha(v)$ | fan-in support reach | $\alpha(v) = \lvert\{k : \mathcal{N}(v) \cap S_k \neq \varnothing\}\rvert$ with $\mathcal{N}(v)$ the fan-in neighbours of $v$, so $\alpha(v) \geq 2$ marks a gate that reads two functional supports (Definition 1) |
| $\tilde{\alpha}(v)$ | aggregation reach | $\tilde{\alpha}(v) = \lvert\{k : \tilde{\mathcal{N}}(v) \cap S_k \neq \varnothing\}\rvert$ with $\tilde{\mathcal{N}}(v)$ the neighbours the aggregation layer reads, so $\tilde{\alpha}$ is the quantity a predictor over that neighbourhood reads (Definition 1) |
| $\mathrm{viol}^{g}_{M_c}(W, X_K)$ | violation count | patterns on which the window's response, read through grouping $g$, differs from the model's output bus (Eq. (3)) |
| $d, w, \eta, \lvert\mathcal{M}\rvert$ | hypothesis space | four counts fixed before evaluation: $d$ operand-group orderings, $w$ output windows, $\eta$ operand-swap and bit-order hypotheses, and $\lvert\mathcal{M}\rvert$ reference models. $\beta \leq 32$ is the bus width of a model |
| $g$ | grouping | the injective map a check uses from a window $W$ onto the output bus of a reference model, enumerated by the search (§3.3) |
| $c^\star(R)$ | proposed class | the class hypothesis a candidate region carries into a round (§3.3) |
| $\mathcal{A} = \{A_1, \ldots, A_m\}$ | candidate list | the finite list of candidate gate sets a run orders, capped at $m \leq 32$ regions per design |
| $\mathcal{A}_t$ | eligible candidates | arms with fewer than three attempts in round $t$ (Eq. (2)) |
| $n_a, h_a, a^\star$ | arm counters and selection | attempts and hits of arm $a$, with $a^\star$ the arm a round selects (Appendix C) |
| $N$ | plays made | the rounds already played, so $N = t - 1$ at the round indexed $t$ of Eq. (2) |
| $L$ | locked set | gates an accepted window has already assigned and locked, which every later revision leaves in place (Algorithm 1) |
| $U$ | unwritten gates | gates whose class every accepted window has left unwritten, so they can still be revised (Algorithm 1) |
| $A_{\max}, B_{\mathrm{tot}}$ | attempt cap and total check budget | three attempts per candidate and at most 2000 checks per design (Algorithm 1) |
| $M_c, \beta$ | reference model of class $c$ | maps an input pattern to the output bus of class $c$, one bit per window output, $\beta \leq 32$ bits wide (Definition 3) |
| $B_{\mathrm{ctrl}}, B_{\mathrm{arith}}, B_{\mathrm{bnd}}$ | per-round role caps | 200, 1200 and 600 checks for a control, arithmetic and boundary candidate (Appendix C) |
| $\mathcal{I}_{ij}$ | interface of two cores | $S_i \cap S_j$, the gates two regions own, and every gate of it has $\kappa \geq 2$ (§4.2) |
| $K$ | pattern budget | random patterns simulated per conformance check |
| $K_1, K_2, K_3$ | pattern budgets (64, 16, 4) | patterns used by the hard check, the $T_2$ soft-confirmation pass and the $T_3$ seed pass (Appendix C) |
| VDR | Verification-Driven Refinement | the refinement stage: bounded, checker-driven class correction |
| UCB1 | upper confidence bound (bandit policy) | orders candidate regions by attempts and hits under a fixed formula |
| PODE | per-output decomposition | splits a design into support cones of its output buses to propose candidates |
| PI / PO | primary input / primary output | netlist boundary nets. Candidates are keyed to multi-bit POs |
| AIG | and-inverter graph | the gate representation used by the equivalence tools |
| SAT | Boolean satisfiability | the engine of miter-based equivalence checking in the efficiency suite |
| cec | combinational equivalence checking | the monolithic ABC baseline that certifies equivalence from the verdict alone |
| MCC | Matthews correlation coefficient | multiclass chance-corrected agreement, used here as a recognition diagnostic |
| mF1 | macro-averaged F1 | unweighted mean of per-class F1, so minority classes count equally |
| bAcc | balanced accuracy | mean per-class recall |
| GNN | graph neural network | family of the learned classifier and of the baselines |
| GraphSAINT | graph sampling and aggregation | the classifier backbone, unchanged across experiments |
| MRF | Markov random field | an ablation arm that smooths predicted classes over the graph |
| TO | timeout | a configuration that reaches the per-check time cap with the verdict pending |

# B Complexity and Scope of Conformance Checking

## B.1 Termination

Each scheduler selection chooses one candidate and increments its attempt counter once, even if several configurations are tested. A configuration specifies a window, a reference model, and the input and output mappings used in their comparison.

**Observation 1** (Finiteness of the procedure)**.** *For a fixed set of $m$ candidate arms, a per-arm attempt cap $A_{\max}$ bounds the number of selections by $T_{\mathrm{sel}} \le A_{\max} m$, and the bound holds while attempt counters stay fixed once incremented. With $A_{\max} = 3$, the bound is $3m$.*

*Proof.* Let $n_a$ count selections of arm $a$. Every selection increases exactly one counter, and an arm becomes ineligible when $n_a = A_{\max}$. Hence,

$$T_{\mathrm{sel}} = \sum_{a=1}^{m} n_a \le A_{\max} m.$$

□

Acceptance writes only to $\mathrm{cone}(W_t^\star) \setminus L_t$ and then adds the accepted cone to $L_t$. Thus, $L_t \subseteq L_{t+1}$, and later updates leave an accepted assignment in place. A newly locked gate may already carry the accepted class, so acceptance leaves its label in place.

## B.2 Cost of Conformance Checking

For configuration $j$, let $s_j$ be the number of gates in the support cone being simulated, $c_j$ the cost of input mapping and reference-model evaluation for one pattern, and $W_j$ the compared output window. With bounded-fan-in gates, evaluating $K_j$ patterns and comparing their outputs costs

$$O(K_j(s_j + c_j + |W_j|)) .$$

For $Q$ tested configurations, the total simulation and comparison cost is therefore bounded by

$$O\left(\sum_{j=1}^{Q} K_j(s_j + c_j + |W_j|)\right) .$$

This bound excludes feature extraction, classifier inference, candidate construction, and search bookkeeping.

The selection bound $T_{\mathrm{sel}} \le 3m$ bounds the selections and leaves $Q$ open, because one selection may test several configurations. Moreover, each test becomes more expensive as its support cone grows. A bounded number of tests therefore leaves the runtime tied to circuit size.

## B.3 Scope of Finite-Pattern Conformance

Fix a configuration $q$ before drawing its validation patterns. Let $F_q(x)$ and $M_q(x)$ denote the circuit and reference output vectors under the specified input and output mappings. Their disagreement probability under uniform primary-input assignments is

$$\delta_q = \Pr_x[F_q(x) \neq M_q(x)].$$

**Proposition 4** (Missed disagreement under random simulation)**.** *For $K$ independent uniform input patterns,*

$$\Pr[F_q(x_k) = M_q(x_k) \textit{ for all } k = 1, \ldots, K] = (1 - \delta_q)^K \le e^{-K\delta_q}.$$

*Proof.* Each pattern produces equal output vectors with probability $1 - \delta_q$. Independence gives the product $(1 - \delta_q)^K$, and $1 - z \le e^{-z}$ gives the upper bound. □

The bound depends on the fraction of inputs that distinguish the two functions. Uniform sampling alone leaves the lower bound on $\delta_q$ open, so $K_1 = 64$ yields a numerical false-acceptance guarantee only through the hypothesis-space count of Appendix C.

A partial window may also agree on a function that two classes share. Without an external carry or borrow input, the least significant output bits of addition and subtraction both equal $a_0 \oplus b_0$, so such a window leaves the class of the region that contains it open. Definition 3 enumerates the groupings at most $\eta$ times per window, and the guard against this degeneracy is the maximum-cone rule of Eq. (4). The rule selects the passing pair of largest support cone, so a window that survives by agreeing on a shared low-order function enters the output only where every larger cone fails.

Gates outside accepted cones retain their classifier labels. This includes gates absent from all candidates and gates whose candidates fail the acceptance test. The conformance bound leaves their classification errors outside its scope.

# C Implementation Details

## C.1 Classifier Configuration

We use the same GraphSAINT classifier configuration across experiments. Each gate has a 46-dimensional descriptor: dimensions 0–33 encode structural features, 34–39 encode topological features, and 40–45 encode simulation-derived features. Evaluation uses standardization statistics fitted on the training data. With $b$ denoting the primary-input count, two descriptor coordinates are transformed as

$$f'_{41} = f_{41} b, \qquad f'_{45} = f_{42} - f'_{41}/32.$$

## C.2 Candidates and Reference Models

We construct candidates from same-class connected components of the predicted field and from support cones rooted at multi-bit output buses. The component-based candidates are split by cosine-radius density clustering of the learned embeddings. Each candidate receives a stable identifier and a proposed class.

The reference library is fixed before evaluation and is constructed from the training pool alone, with reference labels and region masks held out. The listed model families include ripple and carry-chain adders, shift-and-array multipliers, and subtractors and comparators.

For a window $W$ and a reference model $M$, let $\pi$ map circuit inputs to model inputs, and let $g$ map each net in $W$ to a distinct model output bit. The test compares

$$\mathrm{resp}(W, x) \quad \text{with} \quad \big(M(\pi(x))_{g(v)}\big)_{v \in W}.$$

The input mapping specifies operand membership and bit order; the output mapping specifies which result bits are compared.

Among configurations tested within the available budget, we select a passing configuration with maximum $|\mathrm{cone}(W)|$. Ties follow the fixed search order. If every configuration fails, the selection leaves the field in place.

## C.3 Scheduling and Check Budgets

Let $n_a$ be the number of selections of candidate $a$, $h_a$ its recorded hit count, and $N$ the total number of selections already made. The implementation uses

$$u_a = \begin{cases} +\infty, & n_a = 0, \\ \dfrac{h_a}{n_a} + \sqrt{\dfrac{2\ln(\max\{2, N\})}{n_a}}, & n_a > 0. \end{cases}$$

We select the eligible candidate with the largest index and break ties by the smallest stable identifier. Each selection increments $n_a$ once, and a candidate becomes ineligible after three selections. We use this index to order checks and order checks by a fixed formula, with a finite attempt cap in place of a stationary-bandit regret guarantee.

Only configurations passing the $K_1 = 64$ pattern hard check can update labels. After a failure, the $T_2$ tier uses $K_2 = 16$ patterns to guide retries, and the $T_3$ tier uses $K_3 = 4$ patterns to propose seeds. Configurations generated by either tier must pass the hard check before their labels are written.

The configured role limits are

$$(B_{\mathrm{ctrl}}, B_{\mathrm{arith}}, B_{\mathrm{bnd}}) = (200, 1200, 600),$$

with a total check budget $B_{\mathrm{tot}} = 2000$ per design.

**Algorithm 1** Verification-driven class revision

**Require:** Graph $G = (V, E)$, classifier outputs and embeddings, initial labels $\hat{y}^{\text{init}}$, reference library $\mathcal{M}$, attempt cap $A_{\max} = 3$, pattern budgets $(K_1, K_2, K_3) = (64, 16, 4)$, role limits, and total check budget $B_{\text{tot}} = 2000$.
**Ensure:** Labels $\hat{y} : V \to \mathcal{C}$.
1: $\hat{y} \leftarrow \hat{y}^{\text{init}}$, $L \leftarrow \varnothing$, $U \leftarrow V$, $N \leftarrow 0$.
2: Construct candidate arms with stable identifiers and proposed classes.
3: Initialize $n_a \leftarrow 0$ and $h_a \leftarrow 0$ for each arm; initialize the check-budget counters.
4: **while** $U \neq \varnothing$, an eligible candidate remains, and the total check budget has room **do**
5: Select the eligible arm $a^\star$ with maximum $u_a$.
6: $n_{a^\star} \leftarrow n_{a^\star} + 1$; $N \leftarrow N + 1$.
7: $\mathcal{P} \leftarrow \varnothing$.
8: Search the configurations of $a^\star$ within the remaining check budgets. Add each configuration passing the $K_1$-pattern check to $\mathcal{P}$.
9: Use the auxiliary tiers for retry and seed decisions, charging their checks to the applicable counters.
10: **if** $\mathcal{P} \neq \varnothing$ **then**
11: Select $q^\star \in \mathcal{P}$ maximizing $|\text{cone}(W_{q^\star})|$, with ties resolved by search order.
12: $D \leftarrow \text{cone}(W_{q^\star}) \setminus L$.
13: Set $\hat{y}(v) \leftarrow c(a^\star)$ for all $v \in D$.
14: $L \leftarrow L \cup \text{cone}(W_{q^\star})$; $U \leftarrow V \setminus L$.
15: **end if**
16: Update the hit count and candidate search state using the rules specified above.
17: Retire candidates whose attempt or applicable check budget is exhausted.
18: **end while**
19: **return** $\hat{y}$.

### C.4 Label Updates and Algorithm

We initialize the output field with the classifier prediction and maintain a locked set $L_0 = \varnothing$. For a window $W_t^\star$ accepted as class $c_t$, the writable gates are

$$D_t = \text{cone}(W_t^\star) \setminus L_t.$$

The update is

$$\hat{y}^{(t+1)}(v) = \begin{cases} c_t, & v \in D_t, \\ \hat{y}^{(t)}(v), & v \notin D_t, \end{cases} \qquad L_{t+1} = L_t \cup \text{cone}(W_t^\star).$$

The unwritten set is $U_t = V \setminus L_t$. If every configuration fails, we set $W_t^\star = \varnothing$ and leave the field unchanged. Previously locked gates keep their labels even when they belong to a later accepted cone.

### C.5 Scored Field and Evaluation Protocol

Metrics are restricted to the gates that carry a functional ground truth. Every gate of a post-synthesis netlist carries one reference class, because the five class counts of Table 5 sum to the gate count of each netlist, so the scored set is the whole netlist before injection and it is the same at every level of $\rho$. The appended observation gates lie outside every support and are excluded, and the injection leaves the reference class of every other gate unchanged. A class with zero predicted positives has an F1 that is left blank. It appears as "–" and counts as zero in the macro average.

## D Boundary Equivalence-Checking Protocol

We compare each recovered region with its reference core from the clean netlist. The recovered circuit is extracted from the post-synthesis netlist, and both circuits are emitted using the same procedure and an ordered port correspondence.

Let $F_R(x) \in \{0,1\}^p$ and $F_{\text{ref}}(x) \in \{0,1\}^p$ be the recovered and reference output vectors under this correspondence. We construct the miter

$$m(x) = \bigvee_{j=1}^{p} \left(F_R(x)_j \oplus F_{\text{ref}}(x)_j\right).$$

A satisfying assignment to $m(x) = 1$ identifies an input on which the compared circuits disagree. An UNSAT verdict establishes equivalence over all assignments to the declared inputs.

Only an UNSAT verdict certifies the comparison. A SAT verdict, invalid circuit extraction, timeout, or unknown result counts as certification only when the solver returns UNSAT. The verdict applies to the declared output interface and leaves class ownership of shared internal gates to the supports.

## E EVALUATION NETLISTS

Table 5: The evaluation netlists: host design, suite, size, and the three fresh arithmetic cores embedded per netlist (adder, multiplier, subtractor, while control is the host background). *Gates* is the original gate count and equals *Add+Mul+Sub+Cmp+Control*, *Edges* and *PIs* describe that host, and the remaining columns the annotated gates of each class: *Cmp* the comparison logic that sources the gate sharing, *Control* the netlist background. Gate counts are pre-injection and the metrics cover all five classes.

| Netlist | Suite | Gates | Edges | PIs | Add | Mul | Sub | Cmp | Control |
|---|---|---|---|---|---|---|---|---|---|
| Priority | EPFL | 5426 | 8261 | 128 | 198 | 4034 | 236 | 249 | 709 |
| adder | EPFL | 5031 | 7007 | 256 | 194 | 3592 | 233 | 248 | 764 |
| bar | EPFL | 7634 | 12613 | 135 | 194 | 3999 | 233 | 249 | 2959 |
| cavlc | EPFL | 5153 | 8072 | 10 | 194 | 3786 | 233 | 256 | 684 |
| dec | EPFL | 4725 | 7463 | 8 | 194 | 3738 | 233 | 248 | 312 |
| i2c | EPFL | 5834 | 8870 | 148 | 194 | 3975 | 233 | 250 | 1182 |
| int2float | EPFL | 4942 | 7737 | 11 | 194 | 4029 | 233 | 248 | 238 |
| router | EPFL | 4808 | 7474 | 61 | 194 | 3934 | 233 | 247 | 200 |
| c1355 | ISCAS-85 | 4757 | 7412 | 41 | 194 | 3887 | 233 | 248 | 195 |
| c17 | ISCAS-85 | 4554 | 7087 | 5 | 194 | 3868 | 233 | 251 | 8 |
| c1908 | ISCAS-85 | 4684 | 7281 | 33 | 194 | 3762 | 233 | 252 | 243 |
| c2670 | ISCAS-85 | 5147 | 7782 | 234 | 194 | 3831 | 233 | 245 | 644 |
| c3540 | ISCAS-85 | 5502 | 8583 | 50 | 198 | 3883 | 236 | 249 | 936 |
| c432 | ISCAS-85 | 4719 | 7362 | 36 | 194 | 3880 | 233 | 248 | 164 |
| c499 | ISCAS-85 | 4703 | 7298 | 41 | 194 | 3842 | 233 | 248 | 186 |
| c5315 | ISCAS-85 | 6006 | 9095 | 178 | 194 | 3933 | 233 | 248 | 1398 |
| c6288 | ISCAS-85 | 6458 | 10161 | 32 | 194 | 3924 | 233 | 247 | 1860 |
| c7552 | ISCAS-85 | 5926 | 9201 | 207 | 194 | 3989 | 233 | 248 | 1262 |
| c880 | ISCAS-85 | 4864 | 7556 | 60 | 198 | 3916 | 236 | 249 | 265 |
| b10 | ITC-99 | 4446 | 6953 | 38 | 194 | 3762 | 233 | 251 | 6 |
| b13 | ITC-99 | 4086 | 6175 | 74 | 194 | 3410 | 233 | 239 | 10 |

Table 5 lists the netlists: eight EPFL netlist, 11 ISCAS-85, and two ITC-99 netlists (4,086–7,634 gates). Each netlist receives three fresh per-class arithmetic cores with certified functional signatures (zero dropped certification bits per netlist), embedded in the netlist background. The class distribution is imbalanced by design, with the multiplier class dominant. The majority reference and the macro metrics account for that imbalance.

Two scopes must be kept apart. *Embedding* places three arithmetic cores in each netlist. *Recovery and certification* are measured on those same three cores, which carry an annotated region and a formal checker, so the per-case accounting gives $k/3$ for the cores, while the complete task remains the five-class field of Definition 2. The scope is stated with the protocol in Appendix K.1.

## F PER-SUITE RESULTS

Tables 6, 7 and 8 repeat the main grid separately for each suite: all nine baselines and all four of our arms, every metric at every overlap level. The split tests the ordering under a change of netlist family, because the three suites differ in size and in the amount of control logic that surrounds the arithmetic cores. EPFL carries eight netlist, ISCAS-85 eleven, and ITC-99 two, so the ITC-99 block is the one where a single hard netlist could dominate a mean.

The ordering does survive the change of family (Tables 6, 7 and 8). The complete system is first in every suite at every overlap level, and the baseline ordering is stable in all three blocks: DAGNN-RE is the strongest baseline at the heaviest overlap in each suite, at $54.11$ MCC on EPFL, $51.63$ on ISCAS-85 and $48.64$ on ITC-99, while ReVEAL is last in each, at $16.48$, $11.50$ and $8.25$. Because the same ordering appears in three independently constructed benchmark families, the effect holds across a property of one suite.

The height of the majority reference differs between the blocks, and it fixes how much of the score a method obtains from class frequency alone: it is $72.77$ on EPFL, $75.45$ on ISCAS-85 and $84.04$ on ITC-99 (Tables 6–8). The complete system is the only arm whose MCC keeps pace with its accuracy as overlap grows in every block, which is the property that matters when a large share of the gates belongs to the background classes.

The overlap-induced margin at $\rho = 0.75$ is largest on ITC-99, where our system leads the best baseline of that block by $+32.51$ MCC, and it is positive in every suite (Tables 6–8).

## G COST PROFILE AND THE OPTIMIZED ENGINE

Candidate clustering dominates engine time. The timing panel of Table 13 uses a vectorized implementation of that stage together with a faster UCB1 selection, which reach the same class field with

Table 6: EPFL: all methods, all metrics, all $\rho$.

| Method | $\rho = 0$ | | | | $\rho = 0.25$ | | | | $\rho = 0.5$ | | | | $\rho = 0.75$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc | mF1 | MCC | bAcc | Acc | mF1 | MCC | bAcc | Acc | mF1 | MCC | bAcc | Acc | mF1 | MCC | bAcc |
| Majority | 72.77 | 16.78 | 0.00 | 20.00 | 72.77 | 16.78 | 0.00 | 20.00 | 72.77 | 16.78 | 0.00 | 20.00 | 72.77 | 16.78 | 0.00 | 20.00 |
| ABGNN | 96.69 | 91.76 | 93.02 | 93.93 | 87.12 | 57.24 | 68.06 | 57.46 | 81.10 | 33.21 | 49.14 | 35.75 | 77.29 | 27.68 | 40.56 | 32.00 |
| DAGNN-RE | 95.16 | 90.47 | 89.73 | 94.74 | 88.35 | 62.02 | 74.56 | 72.87 | 83.25 | 51.11 | 65.14 | 65.60 | 76.49 | 45.48 | 54.11 | 60.52 |
| GAT | 91.70 | 85.97 | 82.88 | 92.98 | 80.11 | 51.91 | 57.81 | 55.40 | 72.89 | 34.19 | 43.49 | 37.75 | 65.49 | 25.21 | 32.99 | 31.06 |
| GCN | 82.68 | 77.89 | 68.30 | 87.82 | 55.49 | 41.40 | 33.99 | 52.67 | 48.84 | 30.42 | 24.44 | 40.77 | 45.00 | 24.97 | 19.01 | 33.92 |
| GraphSAGE | 92.72 | 88.66 | 85.82 | 94.66 | 75.18 | 52.36 | 54.69 | 61.96 | 54.75 | 34.02 | 34.37 | 44.91 | 34.12 | 23.19 | 19.58 | 33.60 |
| HGCN | 77.76 | 68.59 | 62.19 | 80.58 | 74.48 | 47.17 | 48.64 | 51.59 | 70.76 | 35.33 | 37.46 | 37.65 | 69.19 | 29.83 | 30.40 | 31.80 |
| NeuroMatch | 74.00 | 50.70 | 51.97 | 59.83 | 72.95 | 38.44 | 44.72 | 46.66 | 67.33 | 30.56 | 34.42 | 40.23 | 61.20 | 26.56 | 27.42 | 37.25 |
| ReVEAL | 54.10 | 37.89 | 24.43 | 42.34 | 49.07 | 27.58 | 18.18 | 30.62 | 51.00 | 25.02 | 16.58 | 26.64 | 55.25 | 25.11 | 16.48 | 26.12 |
| WideGate | 92.44 | 85.88 | 84.18 | 92.70 | 83.36 | 55.72 | 61.91 | 58.52 | 78.30 | 37.34 | 46.46 | 38.49 | 75.34 | 28.70 | 38.30 | 32.12 |
| Ours (classifier) | 82.54 | 82.00 | 69.93 | 91.01 | 82.82 | 67.78 | 66.33 | 74.95 | 78.77 | 54.40 | 58.01 | 60.16 | 75.08 | 44.93 | 51.62 | 48.54 |
| Ours (Voting) | 84.92 | 84.49 | 73.20 | 92.19 | 85.69 | 70.64 | 70.78 | 76.66 | 81.19 | 55.72 | 61.07 | 60.00 | 77.17 | 43.71 | 53.40 | 45.70 |
| Ours (MRF) | 83.98 | 84.05 | 71.99 | 91.91 | 84.57 | 69.54 | 69.04 | 76.13 | 80.89 | 55.95 | 60.88 | 61.05 | 77.16 | 45.21 | 53.91 | 47.99 |
| **Ours** | **97.51** | **94.01** | **94.93** | **95.30** | **93.94** | **77.38** | **86.07** | **81.26** | **92.93** | **69.93** | **83.52** | **76.70** | **93.24** | **69.80** | **83.92** | **76.42** |

Table 7: ISCAS-85: all methods, all metrics, all $\rho$.

| Method | $\rho = 0$ | | | | $\rho = 0.25$ | | | | $\rho = 0.5$ | | | | $\rho = 0.75$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc | mF1 | MCC | bAcc | Acc | mF1 | MCC | bAcc | Acc | mF1 | MCC | bAcc | Acc | mF1 | MCC | bAcc |
| Majority | 75.45 | 17.16 | 0.00 | 20.00 | 75.45 | 17.16 | 0.00 | 20.00 | 75.45 | 17.16 | 0.00 | 20.00 | 75.45 | 17.16 | 0.00 | 20.00 |
| ABGNN | 94.95 | 89.15 | 88.90 | 89.94 | 85.52 | 55.71 | 61.60 | 54.55 | 78.82 | 29.90 | 39.07 | 33.74 | 74.63 | 24.07 | 30.42 | 30.24 |
| DAGNN-RE | 93.56 | 88.20 | 85.60 | 91.86 | 87.03 | 62.06 | 69.88 | 71.77 | 82.31 | 52.04 | 60.40 | 65.23 | 77.00 | 46.59 | 51.63 | 60.41 |
| GAT | 90.92 | 85.19 | 79.61 | 90.54 | 80.39 | 51.10 | 54.49 | 53.76 | 72.62 | 31.48 | 37.95 | 35.38 | 64.98 | 23.19 | 27.51 | 29.18 |
| GCN | 81.78 | 78.13 | 63.64 | 85.70 | 57.21 | 42.71 | 32.81 | 52.99 | 50.24 | 30.73 | 24.59 | 39.99 | 45.91 | 24.82 | 20.65 | 33.58 |
| GraphSAGE | 92.79 | 87.28 | 83.65 | 93.19 | 76.03 | 51.47 | 51.94 | 60.79 | 55.45 | 33.68 | 32.51 | 44.42 | 34.11 | 22.60 | 19.87 | 33.18 |
| HGCN | 76.43 | 66.94 | 55.79 | 77.43 | 72.38 | 44.51 | 40.42 | 47.43 | 69.51 | 33.30 | 30.07 | 35.19 | 69.18 | 28.01 | 25.60 | 30.00 |
| NeuroMatch | 76.36 | 49.59 | 47.87 | 57.56 | 75.12 | 39.45 | 42.54 | 45.40 | 70.26 | 33.16 | 35.92 | 40.96 | 64.33 | 29.68 | 30.85 | 38.82 |
| ReVEAL | 54.30 | 38.98 | 20.64 | 44.67 | 49.06 | 26.83 | 14.51 | 31.87 | 50.04 | 23.64 | 12.10 | 27.02 | 54.31 | 23.30 | 11.50 | 25.71 |
| WideGate | 90.17 | 83.11 | 77.72 | 88.36 | 82.56 | 54.67 | 56.53 | 56.08 | 77.17 | 35.53 | 39.56 | 37.33 | 74.02 | 26.18 | 31.58 | 31.11 |
| Ours (classifier) | 84.07 | 81.91 | 68.44 | 89.92 | 82.97 | 67.51 | 62.71 | 73.38 | 78.21 | 53.64 | 52.31 | 58.14 | 73.73 | 42.24 | 43.78 | 45.26 |
| Ours (Voting) | 86.53 | 84.18 | 72.21 | 91.07 | 85.50 | 69.46 | 66.91 | 74.63 | 80.35 | 54.56 | 54.91 | 57.49 | 75.71 | 40.37 | 45.01 | 42.27 |
| Ours (MRF) | 85.67 | 83.75 | 70.89 | 90.80 | 84.58 | 68.64 | 65.40 | 74.26 | 80.04 | 54.60 | 54.79 | 58.61 | 75.76 | 42.31 | 45.95 | 44.76 |
| **Ours** | **96.22** | **92.95** | **92.24** | **93.28** | **92.62** | **76.55** | **82.33** | **79.39** | **91.44** | **69.39** | **79.01** | **75.20** | **91.31** | **67.98** | **78.51** | **74.18** |

fewer conformance checks per case: one at $\rho = 0$ where the full grid of Table 11 spends $44.8$, and 53–55 at the positive levels where the same grid spends $81$. The predicted class field is unchanged on every case of the panel against the stored pre-optimization run.

## H Verification Cost of Certified Boundaries

Every case of this comparison receives a boundary that the design already carries. The $21$ is the recovered multiplier cores taken one per netlist at $\rho = 0$, and the $84$ core instances are the four core families of those $21$ netlists, all measured at $\rho = 0$, because a core artifact is $\rho$-invariant by construction. The $84$ of Table 10 is a different grid, $21$ netlists at four overlap levels. A single-sided algebraic tool supplies the missing side from its own templates, and $^{\S}$ marks the instances for which its own specification applies.

Kissat decides the swept miter on all $84$ core problems in $0.00$ s, which is a preprocessing effect.The swept miter therefore measures what the toolchain does with a congruent instance and not the difficulty of the question. The certification count of Appendix K.6 rests on the plain-miter comparison, where the two sides are different circuits. The swept `cec` lane of the same table is a separate measurement ($63/84$, $3.41$ s). Both sides of that miter are emitted through one emitter over one port list, so the instance is congruent and the structural reduction collapses the two cones as the instance is read. The instance size shows the effect (Table 9). The plain miter has a median of $224$ variables and $690$ clauses and peaks at $2413$ and $13{,}380$, while the swept instance is $122$ variables and $163$ clauses at the median. The plain-miter row on the same instances measures search.

Table 8: ITC-99: all methods, all metrics, all $\rho$.

| Method | $\rho = 0$ | | | | $\rho = 0.25$ | | | | $\rho = 0.5$ | | | | $\rho = 0.75$ | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Acc | mF1 | MCC | bAcc | Acc | mF1 | MCC | bAcc | Acc | mF1 | MCC | bAcc | Acc | mF1 | MCC | bAcc |
| Majority | 84.04 | 18.26 | 0.00 | 20.00 | 84.04 | 18.26 | 0.00 | 20.00 | 84.04 | 18.26 | 0.00 | 20.00 | 84.04 | 18.26 | 0.00 | 20.00 |
| ABGNN | 97.87 | 81.07 | 92.86 | 95.96 | 89.02 | 48.62 | 60.01 | 63.95 | 81.81 | 24.11 | 31.01 | 41.90 | 75.23 | 18.71 | 19.84 | 38.00 |
| DAGNN-RE | 97.21 | 79.26 | 90.89 | 95.46 | 91.13 | 57.51 | 72.86 | 81.17 | 85.45 | 46.53 | 60.86 | 75.36 | 76.90 | 40.68 | 48.64 | 69.45 |
| GAT | 94.20 | 77.12 | 83.68 | 96.69 | 82.99 | 46.20 | 51.43 | 60.95 | 73.51 | 25.61 | 29.73 | 41.48 | 64.70 | 17.97 | 18.78 | 35.80 |
| GCN | 88.46 | 72.30 | 72.28 | 92.94 | 63.65 | 43.25 | 38.64 | 65.54 | 55.66 | 29.29 | 27.45 | 50.89 | 47.51 | 23.39 | 21.56 | 44.09 |
| GraphSAGE | 93.74 | 76.73 | 82.94 | 96.45 | 76.93 | 49.09 | 49.63 | 68.66 | 51.20 | 30.16 | 27.96 | 49.33 | 25.67 | 19.09 | 16.83 | 36.00 |
| HGCN | 81.58 | 62.06 | 60.39 | 84.95 | 76.95 | 41.89 | 41.42 | 58.45 | 73.63 | 31.65 | 32.00 | 47.02 | 72.69 | 25.91 | 27.59 | 42.34 |
| NeuroMatch | 87.15 | 61.20 | 58.95 | 72.30 | 83.47 | 49.60 | 49.47 | 63.60 | 76.73 | 37.34 | 39.62 | 58.65 | 68.62 | 30.99 | 32.29 | 56.04 |
| ReVEAL | 60.55 | 39.41 | 24.18 | 37.06 | 53.98 | 25.69 | 13.60 | 33.21 | 54.63 | 21.25 | 8.91 | 34.88 | 59.49 | 20.49 | 8.25 | 37.05 |
| WideGate | 94.01 | 74.55 | 82.53 | 93.99 | 86.04 | 50.45 | 56.91 | 65.91 | 78.99 | 29.60 | 33.59 | 45.07 | 75.30 | 20.59 | 22.70 | 38.93 |
| Ours (classifier) | 87.77 | 74.73 | 72.33 | 95.36 | 86.02 | 62.73 | 64.40 | 80.54 | 80.17 | 47.62 | 50.46 | 63.72 | 75.11 | 35.46 | 39.89 | 50.37 |
| Ours (Voting) | 89.68 | 76.18 | 75.42 | 96.12 | 88.22 | 63.97 | 68.01 | 81.42 | 82.39 | 47.75 | 52.54 | 62.38 | 77.24 | 33.69 | 40.57 | 47.57 |
| Ours (MRF) | 89.00 | 76.24 | 74.42 | 96.23 | 87.79 | 64.31 | 67.52 | 81.96 | 82.00 | 47.78 | 52.57 | 63.56 | 76.95 | 34.93 | 41.16 | 49.60 |
| **Ours** | **99.50** | **87.81** | **98.24** | **98.24** | **96.31** | **71.23** | **87.40** | **87.16** | **94.83** | **62.20** | **82.52** | **82.01** | **94.40** | **59.63** | **81.15** | **80.50** |

Table 9: Solver performance under the fixed 3600 s timeout and 8 GiB limit. *Solved* counts decisive verdicts against the block size in the heading, so building the miter alone is unsolved. *Avg Time* is the mean over all runs, the next column over decisive runs. *#. fastest* credits the lowest recorded time, ties shared, and a verdict-less row keeps one credit for its miter build. Each lane lists its applicable tools, and a core-lane case is a conditional cost. *Vars / clauses*: for the plain miter a median of 224 variables and 690 clauses (means 706 and 3617, peak 2413 and 13,380), 122 and 163 for the swept instance, a dash where the run left the field blank.

| Tool | Solved | Avg Time (s) | Avg Time, solved (s) | #. fastest | Vars / clauses |
|---|---|---|---|---|---|
| *Core setting, 84 core problems: a boundary is supplied (conditional cost)* | | | | | |
| `cec` | 84/84 | 0.50 | 0.50 | 0 | – |
| swept `cec` | 63/84 | 3.41 | 0.05 | 0 | – |
| Kissat, plain miter | 84/84 | 1.65 | 1.65 | 42 | 224/690 |
| Kissat, swept miter | 84/84 | 0.00 | 0.00 | 84 | 122/163 |
| *Single-side algebraic verifiers on the 21 recovered multiplier cores* | | | | | |
| AMulet 2.0 | 0/21 | 0.03 | – | 0 | – |
| RevSCA-2.0 | 0/21 | 959.0 | – | 0 | – |
| *Same verifiers on a multiplier suite (six designs, four 8-bit, and the two-sided rows cover four of them)* | | | | | |
| `cec` | 6/6 | 0.55 | 0.55 | 6 | – |
| `cec` with `fraig` | 6/6 | 0.20 | 0.20 | 0 | – |
| swept `cec` | 0/6 | 1.29 | – | 0 | – |
| Kissat, plain miter | 6/6 | 1.19 | 1.19 | 0 | – |
| AMulet 2.0 | 4/4[§] | 0.38 | 0.38 | 0 | – |
| RevSCA-2.0 | 4/4[§] | 0.02 | 0.02 | 0 | – |
| *Whole-netlist setting: the post-synthesis netlist is the only input* | | | | | |
| `cec` | 84/84 | 1.93 | 1.93 | 83 | – |
| swept `cec` | 0/84 | 17.21 | – | 1 | – |

The gap between the two time columns of swept `cec` is a tail. The mean over all runs is 3.41 s and the mean over decisive runs is 0.05 s, because 63 of the 84 runs finish in 0.1 s or less. The remaining 21 runs carry about 283 s, so the cost sits in the minority of instances that resist a decision.

AMulet and RevSCA are complete for one pattern class, and each supplies the missing side from its own templates here. The recovered cores are regions of a flattened post-synthesis netlist whose port list and bit order come from the recovery step, so the templates the tools hold leave part of the recovered modules. The verdicts are rejections on 17 of the 21 recovered cores for RevSCA and on all 21 for AMulet. The 0.03 s mean for AMulet is a refusal, while RevSCA spends a mean of 959.0 s before it refuses. Both tools prove their own suite when the specification they hold applies, four of four each in the third block of Table 9, against zero of the 21 recovered cores here (Table 9).

Table 10: Netlist-level equivalence checking on the post-synthesis netlists, with the boundary handed to the tool, under the same 3600 s timeout and 8 GiB limit. *Solved* is over the 84 netlist instances ( 21 evaluation netlists at four overlap levels), and a configuration that only builds the miter counts as unsolved. Within the neutral toolbox, `cec` is the certificate step of the greedy arm. [†]That row leaves the verdict pending, so its fastest-time credit is its miter build.

| Tool | Solved | Avg time (s) | #. fastest | Produces a boundary? |
|---|---|---|---|---|
| *Whole-netlist setting, 21 evaluation netlists at every overlap level (84 netlist instances)* | | | | |
| `cec` | 84/84 | 1.932 | 83 | no |
| swept `cec` | 0/84[†] | – | 1 | no |
| `&cec`, `&fraig` | 0/84 | – | 0 | no |

The netlist comparison appears separately in Table 10, `cec` decides an equivalence property of the entire netlist and returns the verdict alone, so the overlap injection leaves its verdict unchanged and the designed task scores the recovered structure. Its cost is conditional in the same sense as the core lane: `cec` returns the verdict alone, so the last column reads “no” for every configuration and the comparison is unsolved in the unified task of Table 3. The two miter-building configurations build the miter in 0.10 and 0.21 s and leave the verdict pending, and swept `cec` leaves the verdict pending on all 84 instances. The two sides of the check are comparable here because both are emitted through one emitter over one port list, and the same configuration decides 63 of the 84 core problems once a boundary is supplied, in the core comparison of Table 9. A verdict states a property of the netlist, and each of these tools consumes a boundary in hand. Table 3 therefore runs the comparison the other way round, at unknown boundary, where every arm has to produce a boundary under one fixed protocol before the same checker judges it, which is the setting in which a recovered-structure task can be compared at all.

Table 11: Segmented cost over the full grid of netlists at each overlap level, where the five pre-declared netlists are Table 13. Both tables show the same task under two implementations of the recognition engine, whose class fields agree case by case (Appendix G). Aggregate rows are means over the netlists at that $\rho$, instance rows single measurements, and process start-up is excluded. Each stage column and the total column are averaged separately, so a rounded column sum can differ from the rounded total by one unit in the last place. The ‡ marks the level at which one netlist is left out of the mean (Appendix K.6).

| Instance | Suite | $\rho$ | Forward pass | Candidate + Engine | Verify | Total | Checks |
|---|---|---|---|---|---|---|---|
| *Aggregate over the 21 benchmark netlists (means, seconds)* | | | | | | | |
| all benchmark netlists | all | 0.00 | 0.248 | 85.176 | 3.338 | 88.763 | 44.8 |
| all benchmark netlists | all | 0.25 | 0.308 | 38.451 | 1.820 | 40.580 | 81.0 |
| all benchmark netlists | all | 0.50 | 0.392 | 103.842 | 2.060 | 106.294 | 81.0 |
| all benchmark netlists | all | 0.75‡ | 0.431 | 110.474 | 2.408 | 113.315 | 81.0 |
| *Heaviest cases at $\rho = 0.75$, ranked by Total* | | | | | | | |
| c6288 | ISCAS-85 | 0.75 | 0.520 | 163.229 | 3.192 | 166.943 | 81.0 |
| c7552 | ISCAS-85 | 0.75 | 0.493 | 146.648 | 2.312 | 149.454 | 81.0 |
| c5315 | ISCAS-85 | 0.75 | 0.490 | 138.431 | 2.347 | 141.269 | 81.0 |
| Priority | EPFL | 0.75 | 0.468 | 132.451 | 2.560 | 135.481 | 81.0 |
| i2c | EPFL | 0.75 | 0.506 | 132.537 | 2.369 | 135.414 | 81.0 |
| int2float | EPFL | 0.75 | 0.400 | 113.892 | 2.485 | 116.779 | 81.0 |
| b10 | ITC-99 | 0.75 | 0.388 | 55.217 | 2.035 | 57.641 | 81.0 |
| b13 | ITC-99 | 0.75 | 0.333 | 49.842 | 2.057 | 52.233 | 81.0 |

# I Evaluation Protocol and Measurement Accounting

Every method receives the post-synthesis netlist , and the required output is a verified functional structure: the complete node field together with a boundary for every arithmetic core that survives equivalence checking. A saturation solver timed on our own recovered cut is excluded, since the cut is itself the object under test. Equivalence checking natively consumes the whole post-synthesis netlist, so it is timed separately of its own, and because the per-core tools and the single-side algebraic verifiers run only once a boundary exists, and they are recorded as solving the task once a boundary is in hand and their timings are conditional costs measured once the boundary is in hand. Every arm of the solver and equivalence-checking comparisons is measured on one 16-core server, under one 3600 s timeout and one 8 GiB address-space limit, with one process-level resource measurement per invocation, and the times are the method's own computation on an already loaded graph.

A traversal step turns a label field into a candidate boundary for every arm. It reads the field and the probabilities the arm produces, so it reads the label field alone, with reference classes and ground-truth masks held out. Every arm runs under the same cap, the same termination rule and the same judging rule, and a case counts as solved only when all three cores are certified (Tables 3 and 12). No arm is handed a boundary, so the comparison measures the structure each arm's own field produces under one fixed protocol.

## I.1 Scope of the Study

The injected sharing and its levels follow the injection model of Section 4.2, and the class-model library is the assumption every certificate rests on, as Appendix B.3 states. Applying the same propose-then-certify procedure to sequential designs , where register boundaries define a pattern, remains open.

# J Per-Level Ranking of Time to a Certified Result

Table 12 is the per-level view of the same comparison under netlist-level equivalence checking, and it separates speed from capability. The arm the block omits, ABGNN, reaches a finite time in ten of its twenty cases (Table 3). At $\rho = 0$ the complete system solves all five pre-declared netlists at a median of 3.931 s and takes the *#. fastest* count on all five, while the fastest baseline median is 135.56 s, a factor of 34.5. The five times behind that median (Table 13) are 4.269, 3.381, 3.931, 4.534 and 3.448 s, one per netlist in the panel (dec, i2c, c17, c6288, b13). The median and the c17 entry coincide numerically, so the figure is a panel statistic. Four baselines, abgnn, reveal, gat and neuromatch, certify one netlist of five at this level (Table 3) even though they run the same exhaustive toolbox to the same cap, which is the capability difference that the aggregate table hides behind its totals.

The per-level split also shows that the gap narrows while the ordering holds. The ratio falls from 34.5× at $\rho = 0$ to 6.0× at the deepest overlap, because the baselines' cost is dominated by the toolbox, whose search shrinks as overlap makes the problem easier to bound, whereas the system's

Table 12: The designed task ranked per overlap level under netlist-level equivalence checking, at a 300 s cap per case over five pre-declared netlists per block, a case solved only when all three cores are certified. *Avg* and *Median* are over the cases that completed all three cores, *Solved* their denominator, and *#. fastest* charges a failed case as a zero. ABGNN reaches a finite time in 10 of its 20 cases and stays outside the block (Table 3).

| Arm | Solved | Avg time (s) | Median (s) | #. fastest |
|---|---|---|---|---|
| $\rho = 0$ | | | | |
| **Ours** | **5/5** | **3.913** | **3.931** | **5** |
| WideGate | 5/5 | 139.11 | 135.56 | 0 |
| ReVEAL | 1/5 | 137.14 | 137.14 | 0 |
| GAT | 1/5 | 138.31 | 138.31 | 0 |
| GCN | 5/5 | 151.38 | 141.51 | 0 |
| GraphSAGE | 5/5 | 149.01 | 145.22 | 0 |
| HGCN | 5/5 | 151.93 | 146.14 | 0 |
| DAGNN-RE | 5/5 | 151.45 | 146.64 | 0 |
| NeuroMatch | 1/5 | 146.77 | 146.77 | 0 |
| $\rho = 0.25$ | | | | |
| **Ours** | **5/5** | **11.522** | **10.091** | **5** |
| WideGate | 5/5 | 142.21 | 134.17 | 0 |
| HGCN | 5/5 | 150.54 | 140.94 | 0 |
| NeuroMatch | 4/5 | 152.92 | 147.89 | 0 |
| GCN | 5/5 | 149.83 | 148.10 | 0 |
| GAT | 4/5 | 153.76 | 149.27 | 0 |
| DAGNN-RE | 5/5 | 156.11 | 151.83 | 0 |
| GraphSAGE | 5/5 | 158.24 | 153.87 | 0 |
| ReVEAL | 4/5 | 189.66 | 196.02 | 0 |
| $\rho = 0.5$ | | | | |
| **Ours** | **5/5** | **23.984** | **19.051** | **5** |
| WideGate | 5/5 | 141.06 | 133.85 | 0 |
| GCN | 5/5 | 146.95 | 138.17 | 0 |
| HGCN | 5/5 | 151.60 | 144.95 | 0 |
| DAGNN-RE | 5/5 | 152.70 | 144.97 | 0 |
| GraphSAGE | 5/5 | 152.36 | 145.85 | 0 |
| GAT | 4/5 | 153.13 | 148.95 | 0 |
| NeuroMatch | 4/5 | 156.50 | 152.56 | 0 |
| ReVEAL | 4/5 | 180.59 | 187.08 | 0 |
| $\rho = 0.75$ | | | | |
| **Ours** | **5/5** | **23.375** | **21.650** | **5** |
| WideGate | 5/5 | 134.01 | 129.67 | 0 |
| GAT | 5/5 | 148.07 | 140.54 | 0 |
| GCN | 5/5 | 154.34 | 142.00 | 0 |
| DAGNN-RE | 5/5 | 146.28 | 142.87 | 0 |
| NeuroMatch | 5/5 | 149.91 | 144.91 | 0 |
| GraphSAGE | 5/5 | 152.12 | 146.69 | 0 |
| HGCN | 5/5 | 152.92 | 151.69 | 0 |
| ReVEAL | 5/5 | 179.35 | 178.56 | 0 |

cost is the refinement budget, which rises with the number of candidates. The *Avg time* column is computed over solved cases only, while *#. fastest* charges a failed case as a zero. The ordering is therefore one of certified structure. The quantity that separates the arms is the search each of them needs to close a label field into a boundary the checker accepts, and refining the field is the operation that bounds that search (Table 13).

# K Timing, Segmentation and Certification Accounting

## K.1 Classification Metrics

The task of Definition 2 labels every gate with exactly one class, so every prediction is scored against every class and counts as a true negative for the classes it leaves out. The measured metrics therefore reduce to the following. *Accuracy* is the fraction of annotated gates whose single predicted class equals their reference class. *Macro-F1* is the unweighted mean of the classwise F1 scores, each a harmonic mean of that class's precision and recall, so a class the method fails to recover lowers the score even when it holds few gates, and an absent class contributes zero. *Balanced accuracy* is the unweighted mean of the per-class recalls, which removes the effect of class imbalance on the accuracy figure. *MCC* is the Matthews correlation coefficient of the designed field, included because it is the one of the four that a majority-class predictor leaves at zero. All four metrics are *restricted*: only gates carrying a reference class are scored, as fixed in Section 4.1.

All gates carrying a reference class are therefore scored, and that is all five classes: the comparator is the agent of the sharing and the control field is the host background, so both are scored as classes. The injected observation gates of the overlap model lie outside the reference field and are excluded for the same reason (Section 4.2). Node accuracy is a fraction of the *annotated* gates: for each host, the gates the per-netlist counts of Table 5 label with a class, over the five classes of $\mathcal{C}$. The *majority reference* predicts the most frequent class of that set, the multiplier, for every gate, and is aggregated

by host: the per-host accuracy first, then the unweighted mean over hosts, which on the five-class field gives $75.25\%$ accuracy, macro-F1 $17.12$, MCC $0.00$ and balanced accuracy $20.00$. A constant predictor reaches $1/5$ balanced accuracy and zero MCC by construction, which fixes the height a method reaches from class frequency alone. The metric tables average the per-host fractions over the suite, the overlap level, or the whole grid, so an accuracy of $92.34\%$ is read as the mean over hosts of the share of annotated gates that received their reference class. Every method is scored on the same set of every netlist. The classifier itself is trained on the whole field of Definition 2, control gates included, because the host background carries a class label in the training pool, so the two classes are learned, predicted and scored like the others.

### K.2 Certification Measures

The recovery rate, the certification rate and the time to target are defined in Section 4.1, and this subsection records how they are counted here. The *recovery count* of a case is the number of its three arithmetic cores that receive a region, and it is written as *k*/3 so a partial result stays visible. The *certification rate* is the fraction of those cores whose region passes the two-sided equivalence check of Appendix K.6. The Matthews correlation coefficient of the designed field is measured over all netlists at that level, and a dash marks a level at which the field was left blank. A case that stays uncertified appears through its certification rate and carries a defined time to target only once its third core certifies. A range is the spread across the five netlists.

Table 13: Segmented cost of the optimized implementation on the five pre-declared netlists, single-threaded, the case wall clock being **Forward** plus **Engine** plus **Verify**, and **Checks** the conformance checks the case consumed. The last row of each block gives the mean and median over its five netlists, start-up excluded, and the pre-optimization grid is Table 11. Every case returns a certification rate of $1.00$, predicted classes unchanged from the pre-optimization run.

| Instance | $\rho$ | Forward pass | Engine | Verify | Total | Checks |
|---|---|---|---|---|---|---|
| $0$ | | | | | | |
| dec | 0.00 | 0.485 | 0.580 | 3.204 | 4.269 | 1 |
| i2c | 0.00 | 0.534 | 0.744 | 2.103 | 3.381 | 1 |
| c17 | 0.00 | 0.380 | 0.609 | 2.942 | 3.931 | 1 |
| c6288 | 0.00 | 0.573 | 0.908 | 3.053 | 4.534 | 1 |
| b13 | 0.00 | 0.529 | 0.456 | 2.463 | 3.448 | 1 |
| mean / median | 0.00 | – | – | – | 3.913 / 3.931 | 1 / 1 |
| $\rho = 0.25$ | | | | | | |
| dec | 0.25 | 0.461 | 4.571 | 3.747 | 8.779 | 55 |
| i2c | 0.25 | 0.782 | 14.523 | 2.896 | 18.201 | 55 |
| c17 | 0.25 | 0.670 | 4.825 | 3.522 | 9.017 | 55 |
| c6288 | 0.25 | 0.675 | 6.122 | 4.726 | 11.523 | 55 |
| b13 | 0.25 | 0.631 | 6.335 | 3.125 | 10.091 | 54 |
| mean / median | 0.25 | – | – | – | 11.522 / 10.091 | 55 / 55 |
| $\rho = 0.50$ | | | | | | |
| dec | 0.50 | 0.842 | 12.541 | 5.289 | 18.672 | 54 |
| i2c | 0.50 | 1.096 | 35.832 | 4.156 | 41.084 | 55 |
| c17 | 0.50 | 0.825 | 12.657 | 5.569 | 19.051 | 54 |
| c6288 | 0.50 | 1.168 | 18.240 | 5.970 | 25.378 | 53 |
| b13 | 0.50 | 0.843 | 10.794 | 4.099 | 15.736 | 54 |
| mean / median | 0.50 | – | – | – | 23.984 / 19.051 | 54 / 54 |
| $\rho = 0.75$ | | | | | | |
| dec | 0.75 | 1.028 | 16.560 | 4.062 | 21.650 | 53 |
| i2c | 0.75 | 1.235 | 20.355 | 4.204 | 25.794 | 54 |
| c17 | 0.75 | 0.938 | 13.792 | 4.607 | 19.337 | 53 |
| c6288 | 0.75 | 1.346 | 23.923 | 5.985 | 31.254 | 53 |
| b13 | 0.75 | 0.956 | 14.373 | 3.510 | 18.839 | 53 |
| mean / median | 0.75 | – | – | – | 23.375 / 21.650 | 53 / 53 |

### K.3 Measurement Protocol and Resource Accounting

Every method receives the post-synthesis netlist and nothing else, and the required output is a verified functional structure. The protocol, the measurement accounting and the resource measurement are stated in full in Appendix I. Every arm of the solver and equivalence-checking comparisons is measured under one $3600$ s timeout and one $8$ GiB address-space limit on one $16$-core server, while the designed task of Table 3 runs under its own $300$ s cap per case.

### K.4 Stage-Wise Cost Decomposition

Table 13 separates the cost of the optimized system into the classifier forward pass, the candidate-plus-engine step that turns a class field into boundaries, and the equivalence check that certifies

Table 14: Stage-wise cost of the complete system on the five pre-declared netlists, seconds and single-threaded. Stage columns are means over the five cases at that level and sum to the total, whose mean and median are both given. *Checks* is the median conformance checks a case spends.

| $\rho$ | Forward pass | Engine | Verify | Total (s) mean / median | Checks |
|---|---|---|---|---|---|
| 0 | 0.500 | 0.659 | 2.753 | 3.913 / 3.931 | 1 |
| 0.25 | 0.644 | 7.275 | 3.603 | 11.522 / 10.091 | 55 |
| 0.5 | 0.955 | 18.013 | 5.017 | 23.984 / 19.051 | 54 |
| 0.75 | 1.101 | 17.801 | 4.474 | 23.375 / 21.650 | 53 |

them, on the five pre-declared netlists. A *case* is one netlist at one overlap level, and every time in this section is the wall clock of one case, covering all three arithmetic cores of that netlist at that level, so the five pre-declared netlists give five cases per level and twenty cases over the sweep. At $\rho = 0$ the median case is $3.931$ s and the mean $3.913$ s, the per-case engine times span $0.456$–$0.908$ s and the per-case forward pass $0.380$–$0.573$ s, so the remainder is verification. The engine holds a small share of the case: it holds $0.456$ s of `b13` and $0.908$ s of `c6288` at $\rho = 0$. Overlap reinstates the engine: the median case is $10.091$ s at $\rho = 0.25$, $19.051$ s at $\rho = 0.50$ and $21.650$ s at $\rho = 0.75$, against means of $11.522$, $23.984$ and $23.375$ s, and the engine block alone moves from $0.456$–$0.908$ s at $\rho = 0$ to $10.794$–$35.832$ s at $\rho = 0.5$ and $13.792$–$23.923$ s at $\rho = 0.75$.

The work tracks the injection level, while the attempt count stays flat. On the five pre-declared netlists of Table 13 the optimized implementation consumes exactly one conformance check at $\rho = 0$ and $53$ to $55$ across the positive levels, and the count is flat across those three levels, while the pre-optimization grid of Table 11 sits at $81$ checks per case at every positive level. The engine stops when the candidate regions it can accept are exhausted, so the number of selections is bounded by Observation 1. Appendix B.2 notes that this bound leaves the running time open, and Table 13 gives the measured cost.

Against the per-netlist best of the two toolbox variants the margin at $\rho = 0$ is $1.75\times$ to $2.29\times$, on `dec` $7.62$ against $4.269$ s, `i2c` $7.44$ against $3.381$ s, `c17` $9.02$ against $3.931$ s, `c6288` $7.95$ against $4.534$ s, and `b13` $6.85$ against $3.448$ s. The margin is therefore a property of the stage this paper adds: both configurations share the same forward pass and the same class field, so the difference between them is the proposal-and-check loop (Table 13).

### K.5 Time to a Verified Result

The quantity that separates the methods is the time at which the result exists. Each of the nine learned baselines returns a class field, and each reaches a certified result only after the exhaustive toolbox has searched its own field to the cap, which is where their hundreds of seconds go. Equivalence checking is faster still, $1.93$ s on average over all $84$ netlist instances, with every run returning a verdict inside the limit (Table 10), but it decides a property of the netlist and it states a property of the netlist, while the designed task requires the structure. Under the designed-task protocol all nine backbones do finish, and the separation is one of time: $34.5\times$ at $\rho = 0$, then $13.3\times$, $7.0\times$ and $6.0\times$ as overlap deepens.

The remaining time is the cost of the refinement procedure. On the five pre-declared netlists of Table 13 the check count settles at a flat $53$ to $55$ once overlap is positive, and the median case doubles from $\rho = 0$ to $\rho = 0.25$ by $6.16$ s, while the final step adds only $2.60$ s from $\rho = 0.5$ to $\rho = 0.75$. Figure 4 plots certified netlists against time to target for every arm at every level, and its two tails carry the same statement: at $\rho = 0$ four backbones (ABGNN, GAT, NeuroMatch, ReVEAL) certify one netlist of five, and three of them (GAT, NeuroMatch, ReVEAL) recover as overlap increases, while one case (`abgnn` on `dec` at $\rho = 0.75$) exhausts the whole cap with zero certifications.

### K.6 Core Verification Across the Injection Levels

The measure above is taken against the reference core of the clean netlist, which is well posed only while the recovered region presents the same frontier. Under injected sharing the recovered region is a strict superset: the injected nodes attach observation outputs and extra inputs at the core boundary, so a one-sided emission declares outputs outside the driver set of the region, synthesis prunes them, and the miter build fails at every such boundary. We therefore emit both sides through the same emitter over the same port list, the recovered region from the post-synthesis netlist and the reference core from the clean one, which makes the two and-inverter graphs congruent by construction. Across all netlists and all four overlap levels the three arithmetic cores are certified everywhere: $252$ of $252$ three-core attempts (21 netlists $\times$ four overlap levels $\times$ three cores) are proven, so the certificate lane covers all $84$ cases, whereas the timing grid of Table 11 lacks one $\rho$=$0.75$ case (`bar`), excluded from its timing means only. The two-sided emission makes the count a statement about closure of the recovered regions. The solver cost of the same task is the plain-miter lane of Table 9. On

the optimized five-netlist panel the certification step costs $2.103$ to $3.204$ s per case at $\rho = 0$ and $3.510$ to $5.985$ s at $\rho = 0.75$ (Table 13), so overlap costs under $2\times$ in certification against $5.5\times$ in refinement. The larger cost it imposes falls on the recovery that precedes it.

### K.7 Certification Requires Refinement

The cheapest ordering is to certify the classifier's own regions and run the refinement only where certification fails. On the same cases, with the emission step and the checker held fixed and only the field changed, that ordering succeeds on zero cases: the regions the classifier's field induces form open cuts, so synthesis rejects them and the checker fails to build a miter from them, whereas the refined field certifies every case (Table 13). The reason is structural. The classifier's region for the adder contains cases of the multiplier pattern, so the emitted module drives a case from nets outside the region and synthesis abandons it. Its region for the multiplier omits reference nodes, so the module leaves the ports the checker needs beyond its interface. The refinement stage turns a per-node assignment into a boundary that a satisfiability solver accepts.

### K.8 Reaching a Verdict Under the Same Task

The comparison of speed and capability uses one task, one budget, one termination rule and one judging rule for every arm. Each arm searches until every core is certified or the budget is exhausted, and each core is decided by the same two-sided emission and equivalence check. The baselines receive a toolbox that reads their own labels and probabilities with textbook graph operations. Every arm is charged under the same cap and judged by the same checker, so the arms differ in the origin of a boundary, under one verifier. Field quality is Table 1 under one metric protocol, and a single prior isolates the refinement stage in Table 2. A region must be closed forward to the outputs of the core under certification and backward under transitive fan-in, and both sides must be emitted over a port list the two of them can name, before synthesis accepts the modules and the checker runs on an attempt.

The corrected run is then twofold. First, the toolbox is capable: on $180$ cases at the same $300$ s cap and the same three-core criterion, every backbone completes the task on at least $10$ of its $20$ cases and five do so on all twenty, over $60{,}663$ logged attempts of which $486$ regions were certified, $456$ of them inside the $152$ cases that certified all three cores (Table 12). Second, the field holds out with overlap: only the margin changes. Taking the best backbone per level, the fastest rival by level median is WideGate at $135.56$ s ($\rho = 0$), $134.17$ s ($\rho = 0.25$), $133.85$ s ($\rho = 0.5$) and $129.67$ s ($\rho = 0.75$), against the system at $3.931$, $10.091$, $19.051$ and $21.650$ s: $34.5\times$, $13.3\times$, $7.0\times$ and $6.0\times$. The margin narrows because the system's own cost rises with overlap while a label-space tournament holds its cost. Twenty-eight of the $180$ cases exhausted the cap at $0.95$ of its value or above, and one case (ABGNN on `dec` at $\rho = 0.75$, after $14{,}516$ attempts) exhausted the whole budget with zero certifications. The remaining $152$ terminated because they had solved the case. The near-cap cases mark the cost of searching a field. The same cap leaves our own cases finished at $53$ to $55$ checks per case on the five pre-declared netlists of Table 13, because the candidate regions of a refined field are exhausted before the budget is.