IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN XXXX-XXXX, Vol. 0, No. 0, pp. 1–35. DOI:XXXXXXXX

SoK: Long Paper: A Review of IC Logical Reverse Engineering Techniques

Kevin Xu[1] *, Lucas Daudt Franck[1] and Sam Pagliarini[1]

Carnegie Mellon University, Pittsburgh, Pennsylvania

Abstract. In the modern, globalized supply chain for application specific integrated circuits (ASICs), reverse engineering (RE) techniques can be employed for malicious and benign reasons. This survey defines the specific problem of logical RE from a hardware security perspective, examines the earliest RE-adjacent techniques, organizes contemporary RE works by both objective and methodology, and summarizes publication trends and the evolution of logical RE over the years. We review existing techniques, tracing their evolution from manual evaluation and structural analysis to graph theory and machine learning-based solutions. In addition, the survey identifies common trends and evaluation practices, discussing the strengths and drawbacks of the current literature. We also present a set of unique unaddressed problems, highlighting areas that have not been sufficiently explored as well as completely novel problems in ASIC RE. In conclusion, our findings provide a valuable foundation for researchers interested in RE and the future of the field.

Keywords: Reverse engineering · Hardware security · Application specific integrated circuits · Machine learning

1 Introduction

Integrated circuits (ICs) have become an essential component of modern electronics, and the IC supply chain is highly globalized. Its design, manufacturing, testing, and logistics are distributed across worldwide entities. This exposes chips to potential theft of intellectual property (IP) and hardware Trojan insertion by malicious actors, and creates a degree of distrust in the produced chips [RKK14]. It is estimated that IC overproduction and counterfeiting cost US-based semiconductor companies more than $7.5 billion per year [SIA18]. In response, the US government has taken measures to accelerate domestic IC production; for example, the 2022 CHIPS Act allocated approximately $52 billion into semiconductor research and $39 billion into industrial subsidies for US IC fabrication facilities [Sar25]. Clearly, the globalized nature of the IC supply chain engenders doubts about chip trustworthiness and the protection of their IP.

Logical reverse engineering (RE) offers a potential solution to address these concerns. Its ability to confirm IC functionality and underlying structure makes it highly significant for hardware security. However, deriving the functionality of a chip from an unstructured netlist containing only logic gates and interconnections is a challenging task. In response to this challenge, many techniques have been developed over the past decades. The earliest RE techniques focused on building a gate-level abstraction from a delayered chip. Modern logical RE techniques have expanded from the identification of simple subcircuits to the extraction of finite state machines and other higher-level structures.

*Funded by the National Science Foundation Graduate Research Fellowship Program, grant number: DGE2140739

Licensed under Creative Commons License CC-BY 4.0.

1.1 Logical Reverse Engineering

We define logical RE as the process of deriving high-level components from a low-level representation of a chip design, typically a netlist. These high-level components could include specific subcircuits, arithmetic operators, datapath words, registers, and finite state machines, among others. In the vast majority of techniques, the low-level representation is a gate-level netlist of a digital circuit, which consists of instances from a cell library (logic gates) interconnected by wires (nets). It is typically assumed that the logical RE effort is aware of the standard cell library that was used in the design process of the IC.

Logical RE of netlists is different from other types of RE, such as physical RE, RE attacks, security-centered RE, and FPGA RE. Physical RE normally entails extracting a gate-level description from an IC through destructive methods: decapping, delayering, and imaging the chip [TJ09]. We consider it to be the natural preliminary step to logical RE. RE attacks are defined as techniques that *deobfuscate* a design – a terminology typically associated with logic locking schemes [LYZH16, MGT17]. While these attacks may appear superficially similar to logical RE, both the strategies deployed and the end result are completely different. Security-centered RE aims to detect malicious alterations to a design (i.e., hardware Trojan horses or backdoors) [RR22, HBS22]. Although security-centered and logical RE's use cases may overlap, logical RE learns additional netlist information and obtains high-level knowledge of chip components or functionality. FPGA RE and logical RE share the same objectives and operate at a comparable level of abstraction. However, many FPGA RE techniques utilize bitstream manipulation [BSH12], which differs from the more analytical algorithms used in logical RE of ASIC netlists. Netlist-level FPGA RE techniques do not consider the bitstream, but they often search for FPGA-exclusive structural features, such as lookup tables [NVP+23] or hard blocks [MSNG22].

1.2 Logical Reverse Engineering Applications and Overview

Logical RE enables malicious and benign use cases. Examples are:

- In its malicious use case, logical RE may carry out IP theft of a given IC design, enabling the production of counterfeit chips.
- Conversely, in its benign use case, logical RE can ensure a finished chip has not been compromised during the fabrication process, by confirming that the manufactured IC's structure matches the original design.
- Logical RE can also detect IP piracy by extracting the structure of a suspected IC and comparing it to the original [YYNF21].

Given the geopolitical concerns surrounding IC foundries and their trustworthiness, the consistent application of logical RE could restore trust in chip suppliers and reassure domestic designers, companies, and governments. This SoK paper presents a description of the background and taxonomy of RE techniques, a summary of existing logical RE techniques, and a discussion of trends and insights into future research directions. Section 2 provides an overview of early works related to logical RE (precursors to the field) and presents a categorization of current logical RE techniques. Section 3 reviews existing RE techniques, highlighting their methods, comparing and contrasting their approaches, analyzing broader trends among categories, and suggesting possible areas for improvement. Section 4 expounds upon these areas, discussing current deficiencies in the field of logical RE and introducing completely novel problems that could be addressed in future works.

2 Background and Taxonomy for Logical RE Techniques

2.1 Early Works

Early logical RE and RE-adjacent works typically began with transistor-level netlists and abstracted them into a gate-level representation. In contrast, our review prioritizes RE techniques that start from a gate-level netlist and recover higher-level components such as registers or arithmetic units. This difference is shown in Fig. 1. Most early works are not included in the categorizations we will later present. Nonetheless, they remain important for introducing concepts that underlie modern RE, such as circuit graph representations, binary decision diagrams (BDDs), and library matching. Early RE-adjacent techniques mainly focus on two categories of problems: **circuit partitioning** and **subcircuit finding**. In general, both aimed to reduce circuit complexity and improve the efficiency of the design process.

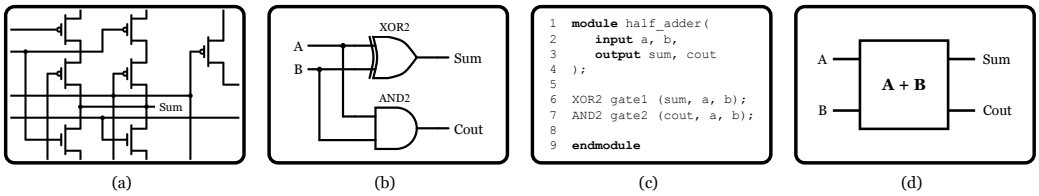


Figure 1: Older RE flow: from transistor-level (a) to gate-level circuit (b). Modern logical RE flow: from gate-level netlist (c) to a detected high-level module (d), e.g., a half-adder.

Circuit partitioning decomposes the given netlist into sets of subcircuits with minimal connectivity. For instance, a technique in [HMV92] converted the netlist into a weighted graph and then applied an eigenvector-based partitioning algorithm, initiating a research trend of eigenvector analysis. Subsequent works used a Tabu Search heuristic [AV93], pre-computed codes [AK96], and a multilevel partitioning algorithm [SLB05]. Other, non-eigenvector based techniques incorporated geometric embeddings [AK93], random walk partitioning [HK92], and analytical placement [RDJ94].

In subcircuit finding, instances of predefined library modules are extracted from a larger circuit. Some techniques addressed the subgraph isomorphism problem, in which a target subcircuit is represented as a subgraph and tested for equality (i.e., isomorphism) against candidate subgraphs in the netlist. An early example, SubGemini [OEGS93], found candidate vertices and matched their surrounding nodes to known subcircuits. SubGemini inspired a wave of subgraph isomorphism-based methods, including bottom-up matching [PR94], genetic representation [VR96], semantic matching [DWWC98, WWCD00], recursive identification scheme [CTL01], fuzzy attributed graph algorithms [ZW03], probabilistic recognition method [Rub03], and directed graph representation [YS03].

Functional features were not analyzed in depth until [EC97], where a gate-level netlist was transformed into logical expressions and BDDs. Despite its pioneering nature, the work consisted of proofs to deduce IC function, signifying a lack of automation. In an RE case study of the ISCAS'85 benchmarks, the authors of [HYH99] combined a simple, automated pattern-matching program for subcircuit finding, with a number of descriptions for manually recognizing certain components in the design. The extensive reliance on manual detection limited this technique's applicability. In contrast, [CELV99] presented an automated tool that created semantic and syntactic definitions of library modules for use in a matching algorithm; however, no experimental results were reported. Another automated technique, CLARION [PBN94], identified registers by clustering flip-flops (FFs) with a common operation (functional feature), or a common linkage to highly connected gates or loops (structural features). As with many early techniques, CLARION faced

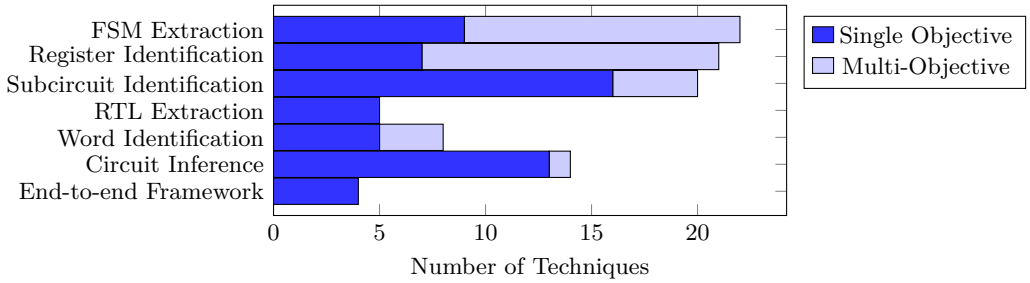


Figure 2: Frequency of works categorized by objective, among the 77 surveyed papers.

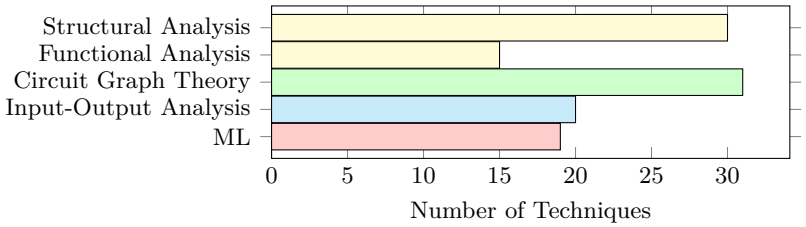


Figure 3: Frequency of works categorized by methodology, among the 77 surveyed papers.

scalability issues even for small sized circuits.

RE of a finite state machine (FSM) was first described in a 2001 patent [McE01]. The method filtered out registers embedded in purely combinational logic or lacking feedback paths, treated the remaining registers' feedback structures as candidate FSM regions, and evaluated different input combinations on Boolean expressions representing the FSM to derive all possible states and next-state functions. Although the assumption of pre-identified register structures was a limitation, many of the concepts introduced here would influence future FSM RE techniques, such as identifying registers with feedback paths, and deriving FSM states with brute-force testing input permutations.

2.2 Categorization Based on Objectives

Existing RE techniques can be classified into seven categories based on their objective(s). **FSM extraction** techniques identify the FSM's associated components from the netlist and/or construct a state transition graph (STG) by exploring the FSM state space. **Register identification** focuses on clustering individual FFs into groups to form registers and/or classifying registers as data/control logic. In **subcircuit identification**, components and functional blocks are found in a larger circuit, either from a pre-partitioned netlist or following the application of a partitioning algorithm. **RTL extraction** techniques aim to recover a (partial) RTL description from a netlist. **Word identification** is defined as detecting word-level structures (operations performed over a bounded array of bits) in a datapath. **Circuit inference** techniques use machine learning (ML) methods to predict the functionality of an entire netlist (full circuit inference) or its subcomponents (subcircuit inference). The **end-to-end framework** category consists of works that assemble multiple tools into a single application. The frequency of each objective category is shown in Figure 2. Although we sought to assign a unique label to each paper, many techniques address multiple objectives simultaneously; therefore, our categorization allows for overlap.

2.3 Categorization Based on Methodologies

The current RE techniques can be divided into five categories based on the methodologies they use (shown in Figure 3). A **structural analysis** method discovers components based

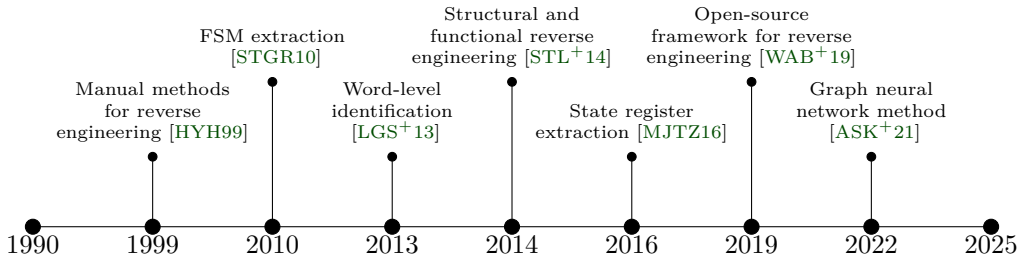


Figure 4: Timeline of RE developments.

solely on circuit structural properties, such as the composition of gate types or arrangement of gate connections. Some prominent subcategories of the method include: *fan-in/fan-out analysis* (analyzing the elements connected to the inputs and outputs of logic instances), *component grouping* (extracting higher-level structures by grouping together wires or FFs), and *library matching* (using a library of known modules and searching for exact matches to their gate structures). A **functional analysis** method examines the logical relationships, Boolean logic, and gate functions of the netlist. It commonly employs *solver-based strategies*, which formulate the circuit as a Boolean Satisfiability Problem (SAT), Satisfiability Modulo Theory (SMT), and/or Quantified Boolean Formula (QBF) and then apply the corresponding SAT, SMT, and QBF solvers to match sub-netlists to modules with known functionalities. Structural and functional methods are usually combined with graph-based representations of the netlist and **graph theory** methods, which comprise an initial circuit-to-graph conversion phase, a partitioning or graph traversal algorithm, and a solver phase that matches subgraphs to known modules.

An **input-output analysis** method repeatedly generates different input permutations, applies them to the IC, and examines the resulting output values to discern information about the IC internal behavior, typically to work out the state space of an FSM. A **machine learning** method learns netlist features and predicts IC modules and functionalities. *Graph neural network (GNN)* methods form a notable subcategory of ML because they are more cohesive in their approach and stated goals. Many papers utilize multiple types of RE methods, resulting in categorization overlaps.

2.4 Evolution of Logical Reverse Engineering

Figure 4 shows a sequence of major developments and important works in the field of logical RE. Some of the earliest automated logical RE techniques were presented in [PBN94] and [CELV99]. In addition, [HYH99] combined manual and automated techniques to RE components of the ISCAS'85 designs. Nonetheless, logical RE did not gain significant research momentum until the development of structural and functional analyses in the 2010s, starting with [STGR10], which introduced one of the first procedural methods for extracting an FSM's structure from a netlist. A wave of logical RE techniques followed, covering the identification of various subcircuit types and notable modules. The RELIC and REFSM tools proposed in [MJTZ16] and [MZJ16a] introduced thorough procedures for register classification and FSM STG extraction, triggering another surge of research on state register and FSM identification. Along a parallel line of research, multiple simulation-based techniques for determining the FSM state space were developed through the 2000s and 2010s, such as in [BSF+09, Smi13]. An open-source platform for logical RE was later developed in [WAB+19]. In the late 2010s, advancements in ML led to the inception of AI-driven logical RE techniques. Early ML-based RE techniques [DB17, FSN+19] mainly applied convolutional neural networks. However, following the publication of landmark tools such as [ASK+21] and [HWB+21], GNNs became the dominant representation for

ML-based logical RE.

To demonstrate the publication trends over time, we consolidate the previously defined methodology-based categories into four macro-groups instead of five (merging Structural and Functional Analyses due to their frequent overlap), in order to make the trends clearer. In accordance with this goal, and unlike Sections 2.2 and 2.3, overlapping categories are not allowed, and each paper is reassigned into the category that best reflects its novel contribution. Figure 5 shows the gradual increase in RE publications triggered by the FSM extraction technique reported in [STGR10], as well as two surges in activity in 2016 and 2022, caused by the "RELIC family" and GNN-based techniques, respectively. The number of papers in each category is displayed as stacked bars in Figure 5.

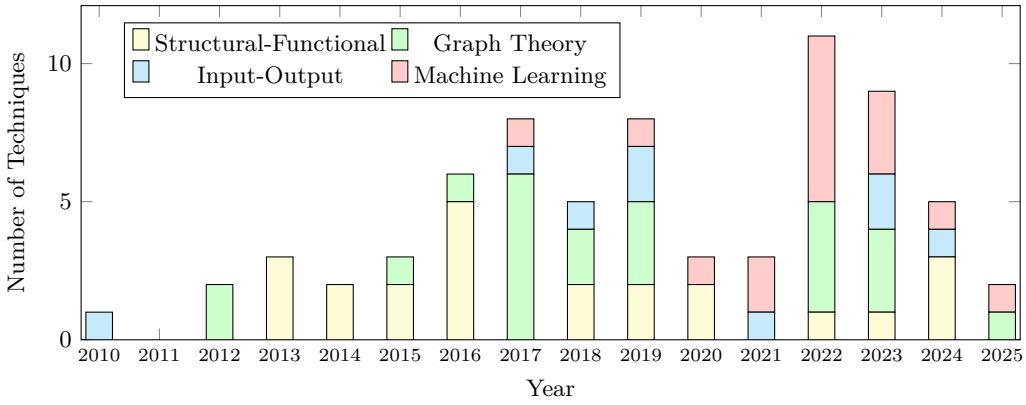


Figure 5: Trend in the number of papers published per category over time

3 Existing Logical Reverse Engineering Techniques

3.1 Word Identification

Word identification techniques focus on extracting bounded bit arrays within a datapath. While this problem is closely related to register identification, the consistent use of the term "word identification" in the literature, along with its stronger emphasis on datapath analysis (as opposed to the sequential, FSM-related analysis of register identification techniques) led to its treatment as a distinct category.

3.1.1 Signal Fan-in Analysis

Li *et al.* [LGS+13] proposed WordRev, one of the first word identification techniques. WordRev grouped wires with similar predecessor structures and functionally equivalent bitslices to form candidate words and then performed symbolic evaluation to propagate words and derive additional candidates. A QBF solver was used to identify the words' operator functions. At the end, the tool produced a data flow graph, allowing users to manually interpret identified words. Although WordRev identified all words from the majority of its benchmarks, it assumed that the bit ordering was known, and required manual interpretation for non-arithmetic operations, which the QBF solver could not classify. Nonetheless, WordRev was a pioneering work in word identification.

Subsequent works drew inspiration from WordRev's usage of fan-in analysis and structural features. Tashjian & Davoodi [TD15] focused on discovering the control signals as a preliminary step. Their procedure substituted values into the control signals, simplified the resulting circuit, and clustered nets with matching fan-in cones into words, repeating the process until no additional words were detected. The technique outperformed WordRev by an average of 10% in detection accuracy, albeit with a longer runtime for large designs, and it struggled to detect state registers. Meade *et al.* introduced the REBUS, REWIND,

and REPCA techniques in [MZJ+16b] and [MSL+18]. Both REBUS and REWIND generated similarity scores for wires and clustered highly similar wires into words. REBUS tracked data flow through the fan-outs of an initial signal set, whereas REWIND selected random "seed" signals and grouped other similar signals with them. REPCA used principal component analysis (PCA) on signals' structural information to group them into words. In addition, Normalized Mutual Information (NMI) was proposed as a common metric for evaluating clustering quality. REWIND attained the highest NMI scores, while REBUS had a superior runtime. However, the effectiveness of all three techniques was sensitive to input parameters, such as the estimated number of words.

3.1.2 Functional Analysis and Large Language Model

In a break from prior techniques, [YC16] implemented a purely functional analysis. The method translated a netlist into the algebraic domain, applied algebraic rewriting, and examined the resulting intermediate pseudo-Boolean expressions to identify words. While the technique successfully extracted arithmetic word-level operations, the evaluated benchmarks were limited to one addition and one multiplication word. An even further divergence came with ReBERT [ZDT25], which was the first and only technique (as of writing) to apply large language models (LLM) to logical RE. Its preprocessing step encoded a netlist's bits and their fan-in cones into token sequences. A custom LLM predicted the probability that bits belonged to the same word. ReBERT demonstrated high word detection accuracy and resilience to "corrupted" circuits (in which random gates were replaced with functionally equivalent structures), but had poor runtime for large benchmarks. While LLMs excel at capturing natural language, it remains unclear whether this capability translates effectively to netlist-based representations.

3.1.3 Challenges and Recommendations for Word Identification

The fundamental challenge of word identification lies in determining which wires are *similar* and grouping them to form words. Among the existing solutions, structural similarity is the predominant heuristic. Most "traditional" algorithmic techniques approximate a wire's structural characteristics through its fan-in cone and compare fan-ins based on their hierarchy (gate organization) or composition (gate types). Only the functional analysis approach in [YC16] marked a clear departure from the fan-in cone paradigm; however, its reliance on converting netlists into algebraic expressions likely limited scalability. Overall, fan-in analysis is a recurring theme across algorithmic logical RE works, particularly in word, register, and subcircuit identification.

Despite its popularity, fan-in analysis brings the challenge of depth selection. Increasing the depth expands the scope of the fan-in cone and enables a more accurate characterization of a wire's structural properties, but at the cost of an increased runtime. Reducing the depth lowers runtime but also decreases circuit coverage. Another challenge is deriving functional meaning from identified words. While they can reveal module boundaries, the sheer size and density of words in large circuits render user interpretation difficult.

In light of this difficulty, we recommend using NMI as a standardized evaluation metric for word identification, as it evaluates the quality of wire groupings (into words) against a ground truth (RTL knowledge of actual word groupings) rather than assess the words' semantic functionality. Given the prevalence of tools that match fan-in cones via gate composition (such as in [LGS+13, TD15, MZJ+16b, MSL+18]), we also encourage the exploration of alternative structural similarity features. In a past example, REPCA used novel attributes, such as a signal's fan-in and fan-out sizes or its proximity to the closest FF. Since its publication in 2018, more advanced ML methods have largely supplanted PCA reduction; ReBERT was the only one to apply advanced ML to word identification, but

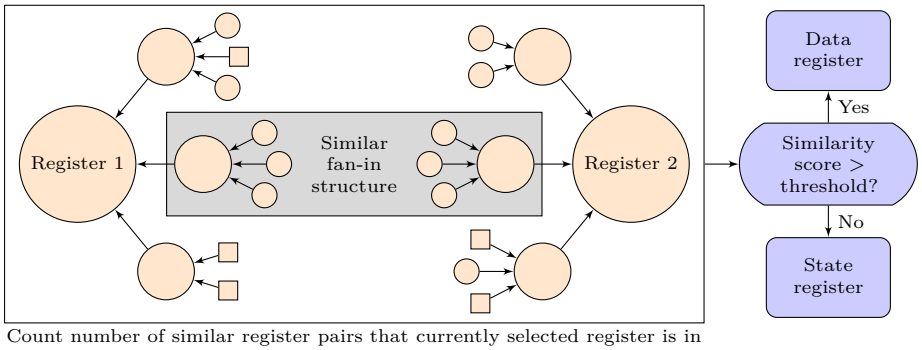


Figure 6: Illustration of the register classification process in RELIC [MJTZ16]. The diagram (left) shows the fan-in comparison for similarity score computation, where shapes represent gate types, and arrows indicate gate hierarchy. The flowchart (right) shows how the similarity score is thresholded to determine if a register is state or data.

its usage of an LLM for netlist representation is likely subpar. In contrast, a GNN-based method would likely be more accurate and promising.

3.2 Register Identification

Register identification techniques typically fall into two categories: state or data classification, and FF grouping. Techniques in the former category analyze predecessor structures of wires or signals, and classify them as part of the control logic (state) or datapath logic (data). Techniques in the latter category cluster FFs into registers, on the basis of common structural features or similarity metrics.

3.2.1 State or Data Classification

Data registers tend to have regularized structures, leading to highly similar fan-in gates, whereas state registers are governed by unique control conditions, leading to more distinct fan-in logic. RELIC was the first technique to capitalize on this pattern [MJTZ16]. For every possible pairwise combination of signals, a recursive algorithm traversed the signals' fan-in structures and computed a similarity score based on the number of gates sharing the same logic type. The similarity heuristic is visualized in Figure 6. The classification scheme assigned each register a counter value, incrementing it whenever the register appeared in a highly similar signal pair. Registers with a count above a certain threshold value were classified as data registers, and those below as state registers.

RELIC was not fully automated; it required a depth parameter for the recursive search, as well as several thresholds for similarity score and register counter calculations. Its runtime was prohibitive for large netlists. Moreover, the similarity score depended solely on a structural feature: whether two logic gates were of the same type. Including functional features could potentially improve the technique's resiliency to structural variance.

Subsequent techniques sought to address these issues. The first follow-up technique, fastRELIC [BBS19], emphasized calculating similarities between register *groups* instead of individual pairs. The algorithm followed RELIC's pairwise combination loop, except registers found to be "similar" were immediately clustered together. With further iterations, the register groups gradually expanded in size, and the number of comparisons in each stage shrunk, thus improving the runtime. A second follow-up technique, RELIC-FUN [GMZJ20], incorporated functional features into analysis. At every register input signal, RELIC-FUN performed a breadth-first search (BFS) on the fan-in, collected k-feasible bitslices, and exhaustively evaluated input permutations for pairs of bitslices. Bitslices proven to be

functionally equivalent were grouped into equivalence classes. Classes with few members were labeled as control logic, while larger classes were labeled as data registers.

These subsequent techniques offered substantial improvements over the original RELIC work. Algorithmic refinement led fastRELIC to achieve up to a 100x speedup, though it still required parameter tuning. RELIC-FUN displayed better resiliency to highly optimized netlists than RELIC and was faster in most benchmarks, but its brute-force equivalence checking resulted in significantly longer runtimes in some cases. The most recent technique in this category, [MBG+24], simulated fault injection and propagation in an IC, compared the simulation results with and without the injected faults, and analyzed the difference to classify FFs as control or data.

3.2.2 Flip-Flop Grouping

Parallel to the line of RELIC-based work, a succession of tools focused on grouping FFs into registers. This trend began with [STL+14], which proposed an algorithm that grouped chains of sequential elements with the same reset/set conditions into registers. Additionally, the work in [FWR+19] grouped FFs sharing a common control signal into register groups, although this step served only as a preliminary stage for discovering combinational blocks. A more detailed discussion of these two techniques can be found in Section 3.3.2.

DANA, the first "pure" grouping technique, was presented in [AHT+20]. This method executed all pairwise combinations of nine different "passes" (grouping methods), splitting or merging FF groups based on control signal or successor/predecessor properties. Pass combinations were iteratively tested until a specific grouping arrangement recurred, at which point DANA halted and selected it as the final register set. In a "steered mode," users could specify an expected register size, leading the algorithm to prioritize groups of that size, and thereby speeding up the runtime. Using the NMI metric from [MSL+18], DANA successfully recovered registers; even in cases with low NMI scores, the register had been successfully detected – albeit fragmented in multiple groups instead of the expected, single register. Compared to RELIC, DANA did not automatically distinguish between control and state registers, so the user had to manually interpret the registers' actual functionality. On the other hand, DANA did not mandate parameter tuning.

3.2.3 Challenges and Recommendations for Register Identification

For state or data classification tools, the main technical challenge is distinguishing registers by their degree of similarity: data registers exhibit repetitive, structurally similar circuitry, whereas state registers are highly dissimilar to other netlist components, reflecting the unique conditions under which they are activated. For FF grouping tools, the main technical challenge is finding a common characteristic that can be used to cluster the FFs. Fan-in analysis is a popular method for state/data classification. RELIC and fastRELIC calculated similarity scores by matching fan-in gate composition, while RELIC-FUN compared bitslices' truth table outputs. A notable difference between the two sets of methods is that bitslices reach their maximal extent at a specified number of inputs, ensuring a consistently sized Boolean expression for functional analysis, while fan-in cones terminate at a predefined depth, ensuring a consistently complex collection of gates for structural analysis. In FF grouping, techniques frequently check the wire connected to the D input signal of FFs to guide grouping; one exception, DANA, uniquely combined multiple grouping heuristics to improve clustering robustness.

A limitation of this control signal-based approach is that synthesis tools may insert buffers or duplicated combinational logic into the wire path to satisfy timing constraints. As a result, FFs that share a common control signal may appear to be driven by different nets at their input pins. DANA merged wire path components to address this problem, but other techniques did not. We urge future techniques to account for the effects of synthesis

optimizations. Returning to the register classification category, the challenge of fan-in depth selection persists. We propose the development of techniques that automatically adjust the depth to attain optimal accuracy; the key challenge here would be runtime optimization, as such adjustments would entail repeated tool executions.

Finally, we recommend classification accuracy as a standardized metric for state/data register identification techniques; ground-truth labels could be readily obtained from annotated RTL descriptions.

3.3 Subcircuit Identification

This category aims to discover pre-defined modules within the netlist. Typically, these modules are combinational circuits and are assumed to be known from a module library.

3.3.1 Pure Partitioning

Pure partitioning forms a small subset of the broader subcircuit identification problem. Techniques in this category partition the circuit into abstract functional "blocks" for later user analysis, without attempting to identify or interpret the subcircuits themselves.

The earliest pure partitioning technique clustered combinational cells within the fan-in cone of a common state cell into subcircuits [Gre13]. The goal was to reduce the number of cells and interconnections, improving netlist readability for subsequent user interpretation. The techniques in [CM18] and [CM19] proposed the problem of reading a high-level, idealized block diagram form of an IC and identifying these reference blocks in the netlist of the manufactured chip. In [CM18], an RTL description of the reference design was assumed to be available. The tool converted the reference and test designs' netlists into weighted undirected graphs, computed geometric embeddings for each node – derived from their eigendecomposition and gate input-output (I/O) features – and applied bipartite matching to label test circuit gates with reference block names. In the follow-up work [CM19], it was assumed the RTL was unavailable, so the *Hclust* algorithm was used to cluster blocks before matching; the rest of the work proceeded similarly to [CM18]. The authors suggested that [CM18] is suitable for designers, with access to a reference RTL design, while [CM19] is suitable for chip buyers. Since neither approach provides an automated means for identifying the blocks' functionalities, their application seems limited to serving as a preliminary step for verification or manual analysis.

In [WBHS22], four established graph-based partitioning algorithms were evaluated for unsupervised clustering of netlists. The NMI metric from [MSL+18] was employed to evaluate partitioning quality. The study concluded that an algorithm named Leiden-CPM had the best performance, in terms of accuracy, runtime, and scalability. However, it required a parameter for the number of expected subcircuits in the IC.

3.3.2 Subcircuit Finding

In contrast to pure partitioning, this category mandates identification of subcircuits' functionality. We define subcircuit finding as follows: given a library of known modules, a tool successively matches these modules against partitioned, candidate subcircuits. Generally, combinational arithmetic circuits are detected rather than sequential ones. Subcircuit finding frequently relies on library matching and graph theory methods.

The technique in [LWS12] generated simulation traces to obtain subcircuits' I/O patterns, represented these patterns as graphs, and matched them against a library of high-level definitions (HLD) using maximum common subgraph detection. The approach in [GSD+14] constructed a template library instead of the more common structural one and applied an SMT solver to match target subcircuit templates with candidate subcircuits. If a match was found, then the algorithm "filled in the details" of the template and

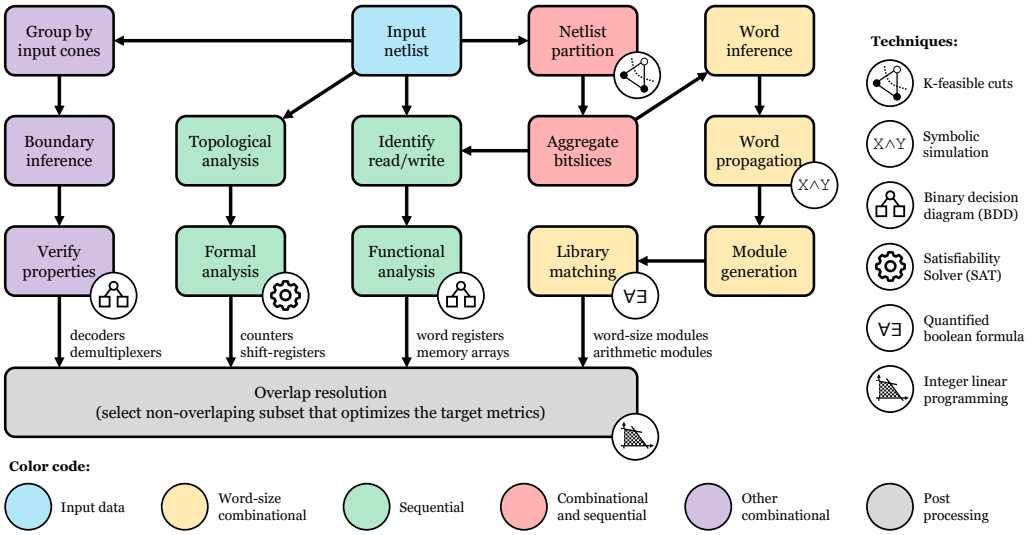


Figure 7: Flowchart showing the various techniques in [STL+14], which is one of the most thorough and widely cited works on subcircuit identification.

synthesized the appropriate mapping of input, output, and control signals. In [SSDB15], simulation graphs were generated for candidate and library subcircuits, allowing for the application of subgraph isomorphism algorithms. None of these techniques included an automatic partitioning step for the input netlists, signifying that considerable manual effort or an additional algorithm would be required.

Several other subcircuit finding techniques did develop their own partitioning algorithms as a preprocessing step. The technique in [SGR+12] converted the netlist into a di-graph matrix and identified combinational blocks via a connected components algorithm, followed by cross matching, I/O signature testing, and BDD matching of library and candidate subcircuits. [STP+13] presented an exhaustive set of techniques that computed k-feasible cuts of gate fan-ins to be bitslices, aggregated bitslices sharing common properties, and extracted circuit segments between detected words and matched them to a library using BDD-based permutation. Sequential components were detected with SAT and BDD-based functional analyses. The follow-up work in [STL+14] enhanced component library matching with a QBF formulation, applied BDD-based functional analysis to detect decoders, and built an integer linear program to resolve overlapping modules. The complete workflow is shown in Figure 7. Assuming XOR trees formed addition columns or carry chains for adders and multipliers, the algorithm in [WDLW15] modeled XOR tree forests as systems of integer linear equations to determine arithmetic module boundaries.

The authors of [MPAR22] developed a "basic algorithm" along with six variants that combined early RE partitioning procedures from [DWWC98] and [WWCD00] to generate subgraph candidates. However, these algorithms struggled on obfuscated benchmarks, which, moreover, were exceedingly small (less than 100 gates).

The approach in [CRSB16] converted the netlist to a graph, employed a normalized cut algorithm for segmentation, and performed fuzzy matching on the resulting subgraphs by comparing their vector embeddings. In [WLBG18], the technique converted the netlist to a weighted graph, where edge weights reflected signal types, and partitioned nodes using the Louvain method, enabling extraction of functional modules (such as the ALU and register files) from cryptographic cores. Furthermore, the method in [FWR+19] initially grouped FFs with the same control signal into register stages and then executed a BFS at each stage to group fan-in gates into bitslices. The authors relaxed subgraph isomorphism to a graph

similarity problem, so their method checked for structural similarity (rather than an exact match) between the library modules and partitioned bitslices. The indirect, fuzzy matching methods employed in [CRSB16] and [FWR+19] represent a meaningful improvement over prior works reliant on exact structural matches, which could be susceptible to small netlist variations caused by defects or synthesis idiosyncrasies.

It is noted in [BHB+22] that open-source designs are exceptionally vulnerable to library matching-based RE. A "golden library" of open-sourced components was rapidly constructed, allowing successful recovery of arithmetic subcircuits, along with a processor and cryptographic core. The most recent case study of subcircuit finding appeared in [PAISR24], which applied RELIC and REPCA to find the Keccak state within a circuit, by detection of FFs as well as analysis of the fan-in and fan-out properties.

3.3.3 Challenges and Recommendations for Subcircuit Identification

There are two major technical challenges in subcircuit identification. The first is partitioning the circuit graph in a sensible manner, such that resultant partitions reflect the structure of actual module boundaries within the netlist. The second is identifying the functionality of these partitions. Early techniques only partially addressed the two challenges: some partitioned the netlist without identifying functionality (such as in [Gre13]), and others identified functionality without partitioning (such as in [LWS12, GSD+14, SSDB15]), assuming the partitioning process was too complex. Later techniques addressed both challenges, by reusing bitslice extraction methods developed for word and register identification (such as in [STL+14, FWD+19]) or by implementing established graph partitioning algorithms (such as in [WLBG18]).

As noted previously, [WBHS22] recommended Leiden-CPM as the most effective circuit graph partitioning algorithm and also highlighted Leiden-Mod, which performs slightly worse but does not require parameter configuration. However, no techniques in the current literature have considered Leiden-CPM or Leiden-Mod, likely due to the recency of the two algorithms (introduced in 2019) and the increasing prevalence of ML-based approaches in the 2020s. We therefore recommend future works apply these recent advances in graph partitioning and integrate them with the functional matching methods from past literature. In addition, the widespread usage of library matching strategies in subcircuit finding introduced multiple challenges, including dependence on library comprehensiveness and sensitivity to structural variations induced by synthesis optimizations. In this context, the fuzzy matching methods described in Section 3.3.2 offer a promising alternative.

From an evaluation perspective, we recommend a "dual," combined metric for future RE techniques: NMI, to assess the accuracy of partitioned subcircuits, and classification accuracy, to evaluate the correctness of identified subcircuit functionalities.

3.4 Circuit Inference

Circuit inference consists of RE techniques that leverage ML to identify circuit functionalities of an entire netlist (full circuit inference) or extract subcircuits (subcircuit inference). Even though the goals of subcircuit identification (Section 3.3) and circuit inference may appear superficially similar, they differ fundamentally in their outputs: subcircuit identification methods produce clearly defined subsets of the netlist, whereas circuit inference techniques generate *predictions* at the gate level or for the entire netlist.

3.4.1 Convolutional Neural Network-based Techniques

Deep learning methods were among the first ML methods applied to logical RE. This trend began with [DB17], which employed a convolutional neural network (CNN). In this approach, the netlist was first converted to a directed acyclic graph (DAG). Each vertex

stored its gate type and an existence vector (EV), which encoded the presence of other gate types in the local neighborhood. A pooling layer selected the "most representative" EVs, transformed them into a Boolean matrix, and inputted matrices to the trained CNN model, which classified entire netlists as multipliers, dividers, or modulo operators. Another deep learning method for RE was presented in [FSN+19]. It was noted that the Boolean matrices used in [DB17] were largely sparse, suggesting unnecessary degrees of freedom. As a solution, the authors used a sparse mapping algorithm to construct EVs, along with level-dependent decaying sum EVs which assigned lower weights to the cells farther from a given vertex. Classes from the previous CNN were expanded to include adders, subtractors, and "unknown operators." As a result, the proposed technique achieved an average classification accuracy about 6% higher than that reported in [DB17].

CNN-based techniques pioneered the application of ML to logical RE; yet, it is unintuitive to transform unstructured netlists of varying sizes into the fixed-size matrix required by CNN input. Additionally, as noted in [ASK+21], the model performance depends on the order in which representative nodes are selected (during pooling), even though the topological order of gates should not influence logical RE results. CNN techniques are also limited to featurizing and classifying an entire netlist at once, rather than automatically identifying subsections, as many prior non-ML techniques do.

3.4.2 K-Nearest Neighbors

The technique in [BBSS20] trained a k-nearest neighbors (k-NN) model on a comprehensive "golden library" of subcircuits, including microcontrollers, interfaces, and memories. Extracted features included gate ratios, the number of inputs and outputs, and circuit graph density. The netlist was partitioned using the Louvain and Markov Clustering (MCL) algorithms, and the resulting subcircuits were classified by the k-NN model, which sustained high accuracy even with partitioning errors.

3.4.3 Hyperdimensional Computing

The core concept of HDCircuit [GASA24b] was to represent an entire netlist as a high-dimensional vector, or hypervector (HV). Initially, a random HV was generated for each gate type. Then, the tool performed a depth-first search (DFS) on every gate, combining HVs of gate types along the search path to form a sum total gate HV; at the end, these gate HVs were bundled into a single HV representing the netlist. To create the ML model, training netlist HVs were bundled together. During evaluation, the model compared a test netlist's HV against the bundled HV for each class, using a similarity metric to determine the final prediction. Full circuit inference experiments recorded an average accuracy of 98%, even on logic locked[1] benchmarks. CircuitHD [GASA24a] described three encoding methods, which differed in their approaches to netlist traversal and hypervector construction: star (DFS on gate neighborhoods), ring (BFS on gate neighborhoods), and path (shortest path between inputs and outputs). Star encoding achieved the highest accuracy at 100%, though resynthesizing the circuit with different technology libraries reduced this accuracy to 75%. IP piracy detection was also discussed as a potential application of this RE technique.

3.4.4 Graph Neural Network

The advent of GNNs enabled ML methods for subcircuit identification and boundary detection. A key advantage of GNNs lies in their ability to learn graph structural relationships at the gate-level and IC-wide scale. During GNN dataset preparation, gates are typically represented as nodes and wires as edges; each node is assigned an initial

[1]Logic locking does not necessarily promote significant changes to the structure of a circuit.

embedding (feature vector). In training, features from each node's neighbors are aggregated to update the node's embedding, allowing it to learn information about its surroundings.

Three techniques published in 2021 marked the start of a surge in GNN works. The first of these, [HLSG21], examined the familiar problem of full circuit inference. Graph convolution layers merged node features (gate types) with graph connectivity information to construct high-dimensional embeddings for entire ICs, allowing the successful classification of adder netlists. The second technique, GNN-RE [ASK+21], represented the netlist as an undirected graph; each node's feature vector recorded connectivity to primary inputs and outputs, fan-in and fan-out degree, gate function (for the node itself and neighbors up to two "hops" away), and whether the gate was sequential. The trained model predicted the subcircuit type to which every gate most likely belonged. GNN-RE achieved very high accuracies in subcircuit and full circuit inference, but a drawback is that its subcircuit inference experiments only tested combinational modules, suggesting a weaker capacity to learn sequential designs. The third technique, ABGNN [HWB+21], identified arithmetic block boundaries. The netlist was represented as a DAG, such that vertices aggregated embeddings from their direct predecessors rather than from a surrounding neighborhood. To capture the entire IC, two separate GNNs performed aggregation in the output and input directions. ABGNN outperformed baseline techniques, [STL+14, WDLW15, FSN+19], but the exact number of detected adders was uncertain: only the percentage of correctly classified nodes was reported.

GNN-RE marked an important advancement as the first technique to perform gate-level classification, setting a precedent for subsequent GNN works. Furthermore, ABGNN's DAG netlist representation and GNN-RE's feature vector selection were often used as a foundation in future GNN techniques. Although these early GNN tools presented a leap in inference capabilities, they relied mainly on structural features, and their accuracy declined for functionally identical but structurally distinct designs.

Subsequent techniques addressed this limitation by integrating functional features into node embeddings and by curating datasets with heavily altered "circuit variants." As an overview, these techniques removed random nodes from select training netlists, simulating circuit approximation ("AppGNN") [BAP+22]; introduced new functional features (truth table and Negation-Permutation-Negation class of a gate's bitslice, as well as signal probability a gate's output assumes a logical value of 1) and rewrote test circuits through input inversion or inverter absorption [ZS22]; used a contrastive scheme in pre-training and replaced random sub-netlists in the test dataset with equivalent Boolean transformations to generate new, augmented netlists ("FGNN") [WBH+22]; devised new functional features, such as if the node was a primary input/output and whether an input edge was inverted ("Gamora") [WLH+23]; employed randomly selected synthesis recipes to produce variants, as well as a loss function to capture the variants' mutually exclusive information, producing similar embeddings for functionally identical designs while maintaining structural invariance ("ConVERTS") [CBC+23]; and integrated structural and functional features from past GNN techniques ([ASK+21, ZS22]) with the bidirectional message passing architecture from ABGNN, resulting in higher F1 scores than the GNN-RE baseline on netlists subjected to logic synthesis optimization [LYL25].

Although effective against circuit variants, most of these techniques focused solely on full circuit inference. So far, only DAGNN-RE, Gamora, and the technique in [ZS22] attempted subcircuit inference. However, DAGNN-RE reused GNN-RE's limited size benchmarks, Gamora targeted a narrow objective (only extracting adders from multipliers), and [ZS22] synthetically generated its benchmarks, which may not have realistic designs.

In a wide divergence from previous techniques, DeepGate sought to learn a neural representation of circuits and framed logic probability prediction as a supervised learning task [LKS+22]. Its GNN incorporated attention mechanisms and reversed propagation layers. A follow-up, DeepGate2 [SPK+23], derived gate truth tables through rapid logic

simulation and added truth table Hamming distance as a node comparison metric, allowing the tool to identify logically equivalent gates within the netlist.

In contrast to the majority of GNN works, which targeted subcircuit or full circuit inference, the objective of ReIGNN and RELIC-GNN was register classification into state or data. ReIGNN [CYN21] added centrality measures to the GNN-RE node features, with the rationale that state registers usually served as "bridges" along shortest paths between nodes. It then deployed Tarjan's algorithm for strongly connected component (SCC) recovery [Tar71], and reclassified any predicted "state registers" that lacked a feedback loop or SCC membership as data registers. An SCC is a maximal set of nodes in a graph, where a path exists from every node in the set to every other node, including itself; these characteristics align with state registers' feedback behavior and connectivity properties. RELIC-GNN [DGP+23] converted registers' fan-in structures into graphs and applied unsupervised learning with a minimization algorithm to generate node embeddings (feature vectors) for these fan-ins. It then adopted the grouping algorithm from fastRELIC, replacing the similarity score with distances between the fan-in feature vectors. Both GNN techniques assumed registers were pre-identified in the netlists. RELIC-GNN achieved higher accuracy than fastRELIC and ReIGNN but required parameter tuning (inherited from RELIC), whereas ReIGNN was fully automated.

3.4.5 Challenges and Recommendations for Circuit Inference

The core technical challenge of circuit inference has been the pursuit of an optimal method for netlist representation, one that could fully harness ML's advantages in combining disparate features and learning subtle patterns. Early works in this category, which used CNN and k-NN models, were effective yet flawed, as they abstracted variable-size circuits into fixed-size matrices. The release of GNNs – ML methods specifically tailored towards learning graph structure – sparked a new wave of research. Initial GNN techniques were reliant on structural features. Later works incorporated functional features, and they augmented training data with circuit variants. As an aside, HDC-based techniques can be viewed as a response to the long training times of neural networks, though they also seek to deliver accurate fixed-size netlist representations, in the form of hypervectors.

An unresolved challenge in GNN techniques is that their classification tasks are typically narrower in scope than those addressed by earlier algorithmic (non-ML) techniques. Although GNN-RE set subcircuit inference as its objective (where a benchmark netlist holds multiple modules to be extracted), most subsequent GNN techniques emphasized full circuit inference (where the benchmark netlist is a single module to be classified). Even GNN-RE's multi-module benchmarks hold at most five subcircuits. In contrast, an older technique such as [STL+14] evaluated designs containing up to 167 adders and subtractors. We thus recommend future GNN techniques evaluate subcircuit inference on more realistically-sized benchmarks. As an alternative direction, future works could combine graph partitioning algorithms with full circuit inference GNN models; the former would extract candidate module structures, while the latter would identify their functionalities. The techniques in [BBSS20] and [DB17] explored a similar idea, but the former relied on a k-NN model (which is not as optimized as GNNs, for the circuit graph format), and the latter implemented an imprecise sliding window method for partitioning.

GNN evaluations could be improved to make results more consistently comparable with other types of tools. For the few techniques that perform subcircuit inference, accuracy is typically reported at the node level, which is not directly comparable with non-ML approaches. We recommend that such tools group nodes with the same subcircuit label (e.g., via a clustering algorithm) and evaluate extracted subcircuits with NMI. For other GNN techniques, we recommend they continue using classification accuracy, as it can be directly compared with the results of non-ML logical RE techniques.

3.5 Finite State Machine Extraction

Finite state machine extraction methods may be divided into two groups: pure input-output behavioral analysis and netlist analysis. Techniques in the former category *only* interface with the circuit (or its simulation view), applying a variety of input permutations and observing outputs to collect state values and transition conditions. Techniques in the latter category operate similarly to other logical RE techniques: they extract the FSM structure directly from the netlist using structural and functional analyses, and may also involve simulation or Boolean analysis of the netlist's functional representation. Construction of the state transition graph is a common goal in both categories.

3.5.1 Pure Input-Output Behavioral Analysis

Since they only interact with the circuit via its primary inputs and outputs, techniques in this category emphasize their non-destructive nature, in contrast to the destructive physical RE methods that extract gate-level netlists. An inherent constraint of this approach is that it cannot identify individual gates or subcircuits. Instead, its strength lies in abstracting the circuit's logical behavior.

The earliest I/O analysis technique applied pseudo-noise generated input sequences to identify I/O pins of FSM blocks and to classify FSMs as "combinatorial or sequential" and as "linear or nonlinear" [BBF+08]. A subsequent technique elucidated distinguishable state transitions by collecting states in a tree data structure, where nodes represented potential states and edges denoted state transitions [BSF+09]. Improving efficiency was the focus of several further techniques, which periodically pruned the state tree [BSF+10], "folded" state tree nodes with identical outputs [Smi13], and optimized the folding process [SOMM17]. Despite a reported 90% reduction in runtime for [BSF+10], the largest benchmark still required a week to finish, while the rest completed in around a day. Later, [Vam18] enhanced state transition identification by combining power side-channel analysis with the tree exploration algorithm of [Smi13]. A follow-up technique represented I/O responses as constraints for an SMT solver [VAV19].

A major advancement in I/O analysis arrived with the realization that scan chains, originally developed for design-for-testability (DfT) purposes, allowed direct observation of register values, reducing the number of required input sequences and improving runtime. Scan-chain techniques read the registers' status at every input change, directly cataloging FSM states [HCC19]; partitioned scan dump data to cluster registers with similar output sequences as FSM or non-FSM registers [CHCL21]; and applied association rules mining techniques to identify FSM state registers [DCL24]. The authors of [DCL24] argued that "traditional" FSM extraction tools face increasing disadvantages: as semiconductor technologies reach the 10 nm regime or below, obtaining the gate-level netlist via physical RE becomes extremely costly and time-consuming. On the other hand, scan-chain-based techniques rely on access to DfT infrastructure, which is not always present; in fact, a simple fuse is enough to prevent unauthorized access to a scan chain [BKW13].

3.5.2 Netlist Analysis

In contrast to I/O behavioral analysis, a separate line of research used traditional logical RE techniques for FSM extraction, applying structural and functional analyses to a gate-level netlist. Techniques in this category are related to register identification, since knowledge of state registers is a prerequisite for recovering the FSM structure. Also, a state exploration step (superficially similar to pure I/O behavioral analysis) is usually employed to construct the STG. Many techniques leverage Tarjan's algorithm for SCC recovery [Tar71], since SCCs share similar properties with FSM state registers and can aid in their identification. [BHBS22] further proposed that FSM recovery techniques could be partitioned into those that harnessed SCCs in their heuristics and those that did not.

We start our discussion with non-SCC-based techniques: [BHBS22] considered state FFs' connectivity and path properties to identify and filter sets of state registers for a "human-readable state machine extraction." These reduced sets were more readable than the full STG, which, when unfiltered, could take the form of a dense, sprawling, and illegible graph. The other non-SCC technique, AVFSM ("Analyzing Vulnerabilities in FSM") [NXY+16], required a synthesis report of FSM state register names[2] which allowed easy recognition of the state FFs. AVFSM inserted XOR and OR gates in front of these identified FFs and wired outputs together, improving coverage of *don't-care* states and FSM state transitions, and then used an automatic test pattern generation (ATPG) approach to obtain the STG. In [BHBS22], the readability, coverage, and accuracy of the resulting graphs depended on the specific FSM and IC design properties, requiring user trial and error to select the best post-processing algorithm for STG abstraction. AVFSM's synthesis report requirement imposed a similar burden on a user or adversary.

SCC-based methods began with [STGR10]. Initially, all registers with feedback paths were treated as state register candidates. A sequence of heuristics then grouped registers that had common enable signals, outputs connected to control signals, and feedback loops forming SCCs. REFSM, [MZJ16a], extended the approach in [STGR10] by generating Boolean expressions from the extracted FSM's gates, and using an SAT solver to explore the state space; the solver results were substituted back into the expressions to compute next states and ultimately produce an STG. However, REFSM was susceptible to XOR trees, whose structures caused inefficient exploration of the full state space. As a response, NETA [MPZJ19], a bundle of RELIC family tools, introduced REFSM2, which identified "superdominator nodes" (areas with massive fan-ins, such as XOR trees), and labeled them as *don't-cares*, thus reducing the algorithm runtime. NETA also introduced several tools to streamline manual RE: REDPEN produced a graph of register dependencies, and RELIC2 incorporated PCA, clustering techniques, and deeper fan-in and fan-out information. A follow-up technique, REFSM-SAT [GMZJ23b], relaxed the FSM extraction problem to identifying next states without determining the corresponding input conditions, thus achieving significantly faster runtime than REFSM. Initially, REDPEN and RELIC were harnessed to find state registers. A new tool, RECUT, extracted combinational logic from the registers' fan-ins, and an SAT solver evaluated the logic to compute next states.

Fundamentally, the REFSM techniques relied on the 2010 FSM extraction tool from [STGR10] to generate candidate state registers. A limitation of [STGR10] is that counters and accumulators fulfill some of its FSM criteria, such as possessing feedback paths and forming SCCs, so they may be falsely identified as FSM state registers.

FSMx prioritized filtering out these "FSM-like structures" [KFFT22]. The method combined state register characteristics from previous FSM extraction works into three conditions for a state-flip flop: its data inputs must be driven by dissimilar standard cells, it must contain a purely combinational self-feedback loop, and it must influence and be influenced by other state FFs. After these criteria were utilized to detect state register regions, a Boolean function analyzer performed brute-force simulation to derive corresponding STGs. A follow-up work, FSMx-Ultra [KFT23], added the SCC detection and common enable signal features from [STGR10]. It employed an optimized version of Tarjan's algorithm with Nuutila's modifications, and extended the Boolean function analyzer with an ATPG approach to refine STG extraction [NSS94].

FSMx reported an approximate tenfold runtime speedup over the baseline method in [STGR10]. We hypothesize that REFSM may attain an even better runtime performance, given its reported efficiency on massive benchmarks and its avoidance of brute-force measures. On the other hand, REFSM did not possess FSMx's vigorous mechanisms for filtering out FSM-like structures such as counters and accumulators, nor did it consider the evaluation of such false positives. Ultimately, runtime remains a challenge in FSM RE.

[2]This information is generally not known by a malicious adversary.

Even in these examples of non-input-output behavioral analysis techniques, heavy-handed functional analysis or simulation is required to fully explore the state space.

3.5.3 Challenges and Recommendations for FSM Extraction

There are two fundamental technical challenges in FSM extraction. The first is recovering the FSM structure itself, in the form of its constituent state registers, and the second is determining the STG, defined by the set of states along with the transitions and transition conditions between them. Existing I/O and netlist analysis techniques strove to meet one, or both, of these challenges, each with distinct advantages and disadvantages. I/O analysis techniques do not require a gate-level netlist or the laborious physical RE process. However, recent techniques often assume scan chain access, which may not be available to adversaries, while earlier scan chain-free techniques suffer from significantly longer runtimes. Netlist analysis techniques, by contrast, are more complex to implement, as they rely on a variety of structural and functional analyses to extract meaningful components from the sea of gates. Both approaches can suffer from runtime issues, since a degree of simulation is typically required to construct the STG.

To evaluate any FSM extraction technique, regardless of approach, we recommend future works follow the example of FSMx-Ultra, in which the ground truth FSM is derived from the RTL and formally compared with the extracted STG using verification tools. For techniques that only recover the FSM, a simple accuracy metric (fraction of correctly identified state registers relative to the total in RTL) would suffice. As a side note, future FSM extraction tools could benefit from combining "traditional" SCC-based methods with the register classification techniques described in Section 3.2.1, which use distinct structural and functional fan-in analyses.

3.6 Targeted RTL Extraction

Tools for converting the gate-level netlist into RTL have only recently emerged. In general, it still seems that logical RE techniques needed to mature before supporting higher-level representations. Even so, the bulk of existing approaches only endeavor for partial RE, producing higher-level descriptions for only a few and select types of circuit components. Some methods are built atop existing RE techniques, and the novelty lies in exporting the results in a different format: namely, RTL code. As a standardized metric, we propose using equivalence checking (e.g., SAT solving) to verify functional equivalence between the extracted RTL and the original netlist. There is also a need for a metric to measure readability. If the generated RTL is too schematic-like, it defeats the purpose.

In the RERTL tool of [PMH+19], a modified version of REFSM was applied. Assuming that logic register connections could be abstracted as signal "dominance", RERTL converted the netlist to a signal graph and applied Lengauer-Trajan's algorithm to identify dominator trees, and thus, the FSM's boundaries [LT79]. REFSM was used to extract the transition logic. Transition conditions were then translated to if-else statements in RTL. Another RTL extraction tool mixed a novel functional analysis with prior simulation techniques [TLH+24]. By analyzing the netlist's input and output vectors, the tool extracted coefficients of conditional and arithmetic operators, transformed random-pattern simulation outputs into a system of linear equations, and applied Gaussian elimination to recover high-level RTL operators such as bit selection, concatenation, and arithmetic operations. The HDL extraction tool in [KEP25] mixed prior techniques from [STL+14] and [STGR10]. Its algorithm converted the netlist to a directed Abstract Syntax Tree, with nodes representing gate I/O terminals and wires. Simple components were identified directly via subgraph matching or targeted simulation. An SCC search on a reduced FF dependency graph then identified FSM candidates, treating SCCs containing 1–100 component vertices as FSMs.

The authors of [SBSH23] and [RS24] minted the term "hardware decompilation" for a new class of techniques. Comparing it to software decompilation, where an executable binary is converted back to source code, the authors described hardware decompilation as transforming a netlist into its original HDL code. In the first of these approaches, a novel intermediate language named "Maki" was developed to linearize the circuit representation, permitting the usage of efficient software clone-detection algorithms for finding repeated Maki code segments [SBSH23]. An HDL "sketch" describing the desired loop code was prepared, and an SMT solver "filled in the holes" with loop and wire variables. In the second hardware decompilation technique [RS24], the tool clustered data FFs with common enable signals and performed DFS traversals to find the correct FF bit order, using topological dependencies. The tool also searched for a memory block by identifying characteristic structural features associated with enable, address, read, and write ports.

3.7 End-to-end Frameworks

So far, only [WAB+19, FWS+19, GMZJ23a] have attempted to develop complete end-to-end frameworks for logical RE. In [WAB+19] and [FWS+19], the authors describe HAL, the first open-source toolchain for RE. HAL featured a graphical user interface for visualizing netlist topology but did not include automated RE tools. Instead, its open-source release enabled users to develop and share RE plugins with the wider community. NetViz, the software from [GMZJ23a], combined the full "RELIC family" of RE tools (RELIC, RELIC-FUN, and REFSM) into a graphical application. NetViz stored input netlists as graphs, displayed them with vector graphics, augmented the view of selected gates with color-coded labeling, and allowed users to select signals as inputs to the integrated tools. [GMZJ23a] proposed a workflow where the users first applied RELIC or RELIC-FUN to identify FSM-related wires, followed by REFSM to explore the FSM state space.

4 Comparative Discussion

Techniques are discussed across multiple dimensions, including runtime and scalability, degree of automation, tool availability, performance metrics, practical deployment challenges, and handling obfuscation and added noise. Representative ones are shown in Table 1.[3] [4]

4.1 Runtime and Scalability

While the wide variety of benchmarks makes objective comparison difficult, some general remarks can be made about the runtime characteristics of various techniques and their associated tradeoffs. Library matching techniques can improve comprehensiveness by adding more module types; however, this directly increases the number of module checks that must be performed for every subcircuit. For fan-in analysis techniques, a parameter typically controls the depth of the searched fan-in structures. An excessively low depth may fail to capture sufficient structure, but increasing the depth invariably increases runtime. The runtime of grouping strategies varies widely depending on the selection of properties to analyze. For FSM extraction techniques, identifying states and state transitions is challenging, as a brute-force, pure I/O behavioral analysis can have prohibitive runtime; even techniques using a measured, Boolean analysis approach may encounter "unsolvable" circuits [MZJ16a]. For ML-based techniques, the dataset annotation and model training processes impose substantial manual effort, particularly for node-level GNN models that assume every gate is labeled. Deep learning techniques, whether CNN- or GNN-based, require tremendous training times, on the order of multiple hours.

[3] In the Time Cost column, O denotes Big-O notation, and n represents the input netlist's gate count.
[4] "ID" denotes "identification," "#" denotes "Number," and "%" denotes "Percentage"

Table 1: Summary of representative logical RE techniques

Reference	Objective	Mean Gates	Time Cost	Availability	Performance Metric	Quantitative Result
[STGR10]	FSM extraction	5330	$O(n)$	Unreleased	# of components	32 FSMs
[LGS+13]	Word ID	58488	$O(2^n)$	Unreleased	# of components	881 words
[STL+14]	Subcircuit ID	19778	$O(2^n)$	Unreleased	Detection %	65% coverage
[TD15]	Word ID	13695	$O(n^2)$	Closed	Detection %	71.89% coverage
[MJTZ16]	Register ID	5368	$O(n^2)$	Closed	Detection %	92.2% accuracy
[CRSB16]	Subcircuit ID	14972	$O(n^2)$	Unreleased	Module comparison	7 matched clusters
[MZJ+16b]	Word ID	2965	$O(n)$	Closed	Outcome-based	Successful pass
[WLBG18]	Subcircuit ID	51898	$O(n \log n)$	Unreleased	Detection %	85% recovery
[BBS19]	Register ID	15497	$O(n^2)$	Unreleased	Detection %	97.5% accuracy
[PMH+19]	RTL extraction	2463	$O(2^n)$	Unreleased	Module comparison	1.515 size ratio
[GMZJ20]	Register ID	4831	$O(n^2)$	Closed	Detection %	50% recovery
[CYN21]	Circuit inference	13072	$O(n^2)$	Open	Prediction %	94.7% accuracy
[HWB+21]	Circuit inference	25082	$O(n)$	Unreleased	Prediction %	95.6% accuracy
[ASK+21]	Circuit inference	4234	$O(n)$	Open	Prediction %	98.9% accuracy
[WBH+22]	Circuit inference	25082	$O(n)$	Open	Prediction %	94.2% F_1 score
[LKS+22]	Circuit inference	32140	$O(n)$	Open	Prediction %	0.0204 error
[BAP+22]	Circuit inference	17583	$O(n)$	Open	Prediction %	86.1% accuracy
[KFT23]	FSM extraction	16579	$O(n^2)$	Unreleased	# of components	66 flip-flops
[GMZJ23b]	FSM extraction	10597	$O(n^2)$	Unreleased	Timing measure	99% speedup
[DCL24]	FSM extraction	87157	$O(n^2)$	Unreleased	Detection %	95% recovery
[SBSH23]	RTL extraction	300	$O(2^n)$	Open	# of components	28 loops rerolled
[LYL25]	Circuit inference	89254	$O(n)$	Unreleased	Prediction %	97.8% F_1 score
[ZDT25]	Word ID	13695	$O(n^2)$	Unreleased	Module comparison	0.35 ARI

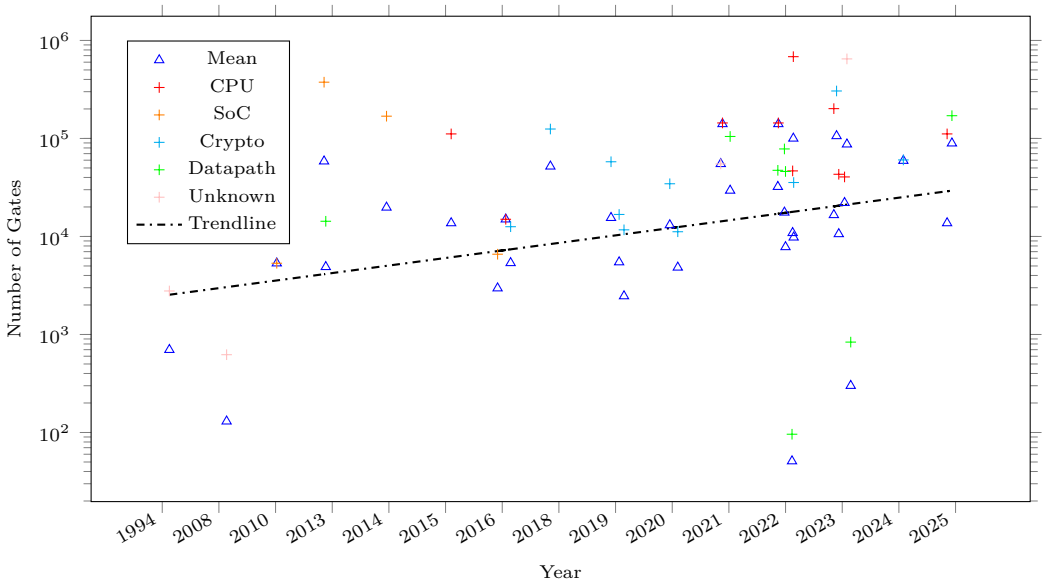


Figure 8: Change in mean (triangle markers) and maximum (cross markers) benchmark sizes over time (measured in number of gates) evaluated in reported techniques. The maximum size points are color-coded according to the category of their design type.

Logical RE techniques typically use designs from OpenCores, secworks, ITC '99, and ISCAS'85 and '89 circuits [Oli25, Str25, Dav99, BF85, BBK89]. The widespread usage of these open-source benchmarks is beneficial, as it ensures direct comparison against past works. However, with the rapid evolution in design trends and IC sizes, these benchmarks may not be representative of modern ICs. While most techniques (among the 36 papers that reported benchmark size) evaluated designs in the range of tens of thousands of gates, only two techniques evaluated designs with at least 500,000 gates, and none exceeded 1,000,000 gates. Average benchmark size has increased over time (see Figure 8), but future logical RE methods must further advance scalability to reflect industry-scale designs.

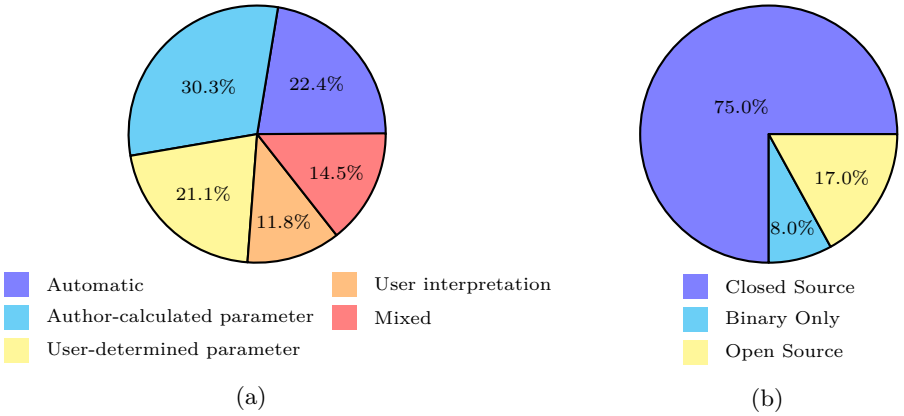
We identify two fundamental barriers to such scaling. One is the paucity of benchmarks with millions of gates, due to the age of the most well-known benchmarks, and the reluctance of chip design companies to release their proprietary IPs. We recommend future techniques source benchmarks from recent open-source designs, which better reflect modern design practices, e.g. OpenTitan [Die25]. The other barrier is high runtime complexity. As shown in Table 1, the majority of selected techniques are estimated to be $O(n^2)$ complexity or worse. Although ML techniques usually have $O(n)$ execution time, they require a prolonged dataset preparation and training time. To overcome the limit of $O(n^2)$ runtime complexity, we recommend that future researchers partition the netlist, using an algorithm such as Leiden-CPM [WBHS22], and apply their RE technique separately on the partitions. Another direction could be "filtering" the netlist for the most distinctive, relevant circuitry (such as control logic) and analyzing this circuitry in the logical RE algorithm.

4.2 Degree of Automation

We define five degrees of automation. **Automatic** denotes tools that run independently, without any form of input or user guidance. **Author-calculated parameter** refers to tools that need certain input parameters, whose optimal values were predetermined by the authors. **User-determined parameter** indicates that the parameter values depend on the input design's properties, requiring user fine-tuning to attain optimal performance. **User interpretation** applies to techniques that explicitly require the user to further analyze results to derive higher-level meaning. **Mixed automatic and manual** describes techniques that combine automated processing with manual steps; we observed that none of the tools required fully manual operations. As shown in Figure 9a, only 22.4% of the published techniques are fully automated, forming less than a third of the 77 categorized publications. Parameter-related levels of automation (author- and user-defined parameters) constitute the bulk of the reported techniques, with "Author-calculated parameter" being the most common. This trend is largely driven by GNN methods, where it is standard practice to list model hyperparameters such as the number of training epochs, learning rate, weight decay, batch size, and so on, for replication of the optimally trained model.

4.3 Tool Availability and Practical Deployment Challenges

Out of the 77 surveyed publications, only 19 released a tool to the public. Among these 19, four tools are from the RELIC family. Also, two publications deal with HAL, reducing the count to 14 unique released techniques. Of these 14, half are GNN works, and the remaining seven consist of [TD15, SBSH23, FWS+19, AHT+20, MPZJ19, GMZJ23a, PAISR24]. Figure 9b shows that the vast majority of all RE techniques are completely closed-source; smaller subsets release executables (binary only) or provide full open-source code. NetViz was the first to distribute binaries for "RELIC-family" tools, although its source code was kept closed-source. HAL was one of the first major logical RE frameworks released as open-source, but it required users to download third-party plugins for automated RE. Otherwise, users must manually examine the netlist. The recent surge in ML-based techniques



(a) (b)

Figure 9: (a) Distribution of tool automation levels in surveyed papers (N=77). (b) Distribution of the proportions of tool availability.

(particularly GNNs) has introduced a trend toward open-source releases. Nevertheless, open-source tools still represent only a small fraction of the overall logical RE literature.

We consider two important practical deployment challenges: computational resources and synthesis tools. Out of the 77 surveyed techniques, 40 reported CPU specifications in their experimental setups. Most used commercial processors (Intel Core i5/i7 and mid-range Xeon). This suggests that many logical RE techniques can be feasibly implemented on a commodity desktop or workstation CPU. However, this assumes access to the tools and benchmarks in the first place; as discussed previously, RE tool availability remains a significant barrier. An additional obstacle to practical deployment is synthesis tool availability; among 29 techniques that reported synthesis details, 18 used commercial synthesis tools (such as Synopsys Design Compiler or Cadence Genus), limiting full reproducibility to industry or academic institutions with appropriate licenses.

Nearly all logical RE techniques noted that synthesis optimization removed hierarchical boundaries (as expected) and impeded netlist analysis, but most did not discuss optimizations beyond listing it as an initial challenge. An exception to the rule was AppGNN, which conducted logical synthesis on training data using the *compile_ultra* directive for area optimization, and outperformed past GNN-RE models on heavily optimized netlists [BAP+22]. Furthermore, RELIC-FUN re-synthesized its benchmarks with an open-source tool and outperformed RELIC on them, showing resiliency to structural variances caused by synthesis optimizations [GMZJ20]. In contrast, CircuitHD's best encoding method suffered from an accuracy drop on netlists resynthesized with a different technology library [GASA24a]. In light of the small number of such exceptions, varying the synthesis tool type and synthesis parameters is a promising area for future exploration.

4.4 Evaluation Metrics

We define seven categories of evaluation metrics. **Number of components found** measures how many desired modules a logical RE technique successfully identified from a gate-level netlist, including number of subcircuit types, registers, words, and states and state transitions. **Clustering measures** consist of NMI and purity, proposed as standardized RE metrics in [MSL+18]. **Timing measures** consist of runtime and speedup. **Prediction percentage** refers to a tool's accuracy in predicting correct components and includes inference accuracy, sensitivity, precision, recall, F1 scores, and counts of false negatives and false positives. **Detection percentage** measures the fraction of desired components identified, or overall coverage. **Outcome-based metrics** are binary "pass/fail" measures. **Module comparison** metrics compare extracted module properties with known modules, such as

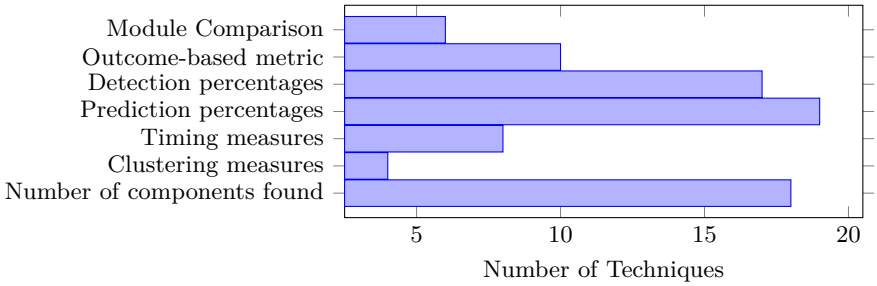


Figure 10: Frequency of each metric type.

Table 2: Summary of recommended standardized metrics

Category	Proposed Standardized Metric
Word identification	**NMI**
Register identification	**NMI and classification accuracy**
Subcircuit identification	**NMI and classification accuracy**
Circuit inference	**NMI and classification accuracy**
FSM extraction	**Formal verification**
Targeted RTL extraction	**Formal verification**

cosine distance between embeddings or similarity scores. Figure 10 shows the frequency of each metric category. *Prediction percentages* is the most common category, likely due to the surge in ML techniques, combined with past non-ML techniques that compared results against the expected number of modules. However, there is a clear issue here: standardized, RE-specific metrics are missing. As a result, papers within the same objective category can only occasionally be compared with prior work. Our proposed standardized metrics are shown in Table 2.

4.5 Handling Obfuscated Circuits

Most methods addressing RE of obfuscated circuits tend to break (or unlock) an obfuscation scheme rather than extract knowledge from the netlist. Typically, the outcome is a deobfuscated circuit, while internal components and their functionalities are left unknown. We consider such works to be attack techniques, separate from the logical RE techniques covered in this survey. Nevertheless, several logical RE techniques included obfuscated designs in their experimental benchmarks. XOR/XNOR logic locking was the most common obfuscation method, as seen in [ASK+21, GASA24a, CBC+23, BBSS20]. ConVERTS [CBC+23] combined XOR/XNOR logic locking with Truly Random Logic Locking [LKK+20] and ASSURE [PCS+21]. Furthermore, the technique in [BBSS20] applied random logic locking with key gates, Strong Logic Locking, and Minimization of Signals with Low Controllability. Despite obfuscation, these logical RE techniques successfully pursued their component identification. As noted in [BBSS20], however, XOR/XNOR logic locking is considered a *basic* form of obfuscation. Still, these techniques mark a significant advance over prior attack methods that did not extract higher-level modules.

More complicated obfuscation methods were explored in [FWR+19], which performed conditional stuttering and sweep transformations. Obfuscated and unobfuscated designs shared a high similarity value, yet no further functional extraction was pursued. In ReBERT, several circuits underwent *corruption*, where certain gates were replaced with functionally equivalent gate sets [ZDT25]. Compared to RELIC as a baseline, ReBERT scored higher ARI across all corruption levels. This generic corruption was more representative of physical RE glitches than genuine circuit obfuscation. In the future, a combination of more complex obfuscation schemes with evaluation tasks equally as rigorous as the ones used for unobfuscated designs would be a valuable direction for logical RE research.

4.6 Handling Circuits with Added Noise

The vast majority of logical RE techniques assumed input netlists lacked glitches (incorrect gates or wire connections from physical RE errors). However, a few RE techniques, primarily ML-based, introduced subtle netlist alterations to simulate physical RE glitches, thereby testing the model's resiliency to defects in the training data. Some techniques directly modified individual gates. AppGNN removed random nodes from circuit graphs, simulating the "approximate circuits" paradigm. ReBERT replaced random gates with functionally equivalent ones, in circuit "corruption." Other techniques operated on groups of gates. FGNN replaced random sub-netlists in the training data with functionally equivalent but topologically different Boolean transformations. ConVERTS applied synthesis "recipes," consisting of algebraic transformations that balanced the circuit graph, gate substitutions that replaced nodes with "shallower" equivalents, and replacements of large sub-networks with optimized subgraphs of fewer nodes. All of these techniques reported strong classification accuracies. However, only AppGNN and ReBERT evaluated their models on noisy circuits and compared their performance to baseline tools (GNN-RE and RELIC, respectively). They reported significant accuracy improvements under added noise, whereas baseline tools struggled. Overall, these four techniques still constituted a tiny portion of the literature. Future works could aspire to add noise to benchmark circuits, improving resiliency to physical RE-induced glitches.

4.7 Unaddressed Problems

4.7.1 Identification of Non-Functional Characteristics

Logical RE techniques have aimed to extract concrete components, such as arithmetic operators, registers, FSMs, etc. Techniques that recover FSM states and transition conditions attempt to extract slightly higher-level information, but the results are still a rearrangement of netlist components (wires and gates) into a concrete, larger structure.

Extracting abstract, less obvious characteristics of an IC remains an unaddressed problem. Examples of such non-functional characteristics of a circuit include timing and power – properties that have not been set as the end goal of any logical RE technique. For timing, in order for an IC to be truly reproduced, characteristics such as false paths, multi-cycle paths, and circuit design margins have to be understood. In short, any timing consideration that an IC designer encodes in a Synopsys Design Constraints (SDC) file is also of interest for RE. Addressing this problem requires simulation-based methods (potentially drawing from the FSM extraction field), along with efficient strategies for exploring the netlist and its timing paths. Perhaps statistical methods that look at timing distributions within a single netlist could bin paths into false, single-cycle, or multi-cycle, but no existing work appears to have attempted this.

4.7.2 Identification of Synthesis Tool

Determining which synthesis tool produced the netlist is another unaddressed problem in logical RE. If a gate-level netlist could be associated to the synthesis tool that generated it, the accuracy of component identification for existing techniques could be increased. Current techniques extract only gate type or wire connection information from the netlist.

This is not an easy problem by any metric. While logic synthesis plays a major role in determining which gates appear in a netlist and how they are interconnected, physical synthesis (i.e., place and route) can modify gates through restructuring optimizations. Furthermore, any naming patterns associated with wires or gates cannot be relied upon for RE purposes: netlists generated from physical RE methods are completely blind. Another motivation for identifying the synthesis tool arises from sanctions or embargoes. Semiconductor technology is a commodity that is often linked to national security, and

countries can restrict access to certain technologies. RE netlists could be analyzed to determine if a design violates these imposed sanctions.

4.7.3 Identification of RTL authorship: Human vs LLM

Recent breakthroughs in AI, particularly in LLMs, have produced models with strong reading, writing, and programming capabilities. "Vibe coding" refers to a new form of software development in the era of AI, where a developer provides high-level design intent through prompts, while the AI performs much of the implementation, translating abstract ideas into actual code. The success of LLMs in automating software development is also extending to hardware development at an accelerated pace.

Detecting if a design was generated by AI will become an important problem. One possible approach is to examine recurring idiosyncrasies in the AI's coding style. Featurization of the code would be necessary, but it is still a challenging and mainly open problem; the tokenization procedure in [ZDT25], or variants thereof, could provide a starting point. However, it is currently difficult to distinguish natural language AI text from human-written text, and a similar level of difficulty can be expected for RTL code. It is completely **unknown** if AI use at design time can be identified in a gate-level netlist.

The RTL authorship problem, although elusive, has real consequences related to IP ownership. Imagine a scenario where a company A buys a 3PIP from company B, only to later discover that it was AI-generated. The IP comes in a netlist format (firm IP) and cannot be easily scrutinized by company A. Not only is the quality of the IP debatable, but its origin is also uncertain. If company B trained an LLM with IP that belongs to a company C, company A could be exposed to liability if it sells a product containing the AI-generated 3PIP. This is just one scenario in what can be a complicated IP ownership landscape for the near future. We envision that RE can play a role, but the technical solutions required to do so remain largely unexplored. A considerable amount of effort would be needed for generating and labeling AI-generated RTL, and their netlists.

4.7.4 Full Reconstruction of RTL

As observed before, existing RTL RE techniques usually extract specific components of a netlist, mirroring the trend of traditional RE techniques that also narrow their focus to individual component types. In the context of RTL extraction, the limited objective is more noticeable, as the output code is necessarily incomplete. A full HDL description of a netlist would be the ideal outcome but is, at present, unfeasible. The underlying issue is simple: netlists carry none of the designer's intent as expressed by identifiers. For example, an RTL variable named *enable* carries a clear semantic meaning, but in a netlist it might become *wire1234*. Although some netlist constructs can be matched to HDL counterparts and given a meaningful identifier, the scale and complexity of modern designs have made such an endeavor almost impossible.

Nonetheless, it would be compelling to see a tool that unifies multiple previous RTL and HDL RE techniques, similar to what [STL+14, MPZJ19, GMZJ23a] accomplished for the non-HDL side of RE. The tool in [KEP25] partially addresses this problem, but its description lacks detail, as the tool was patented. AI could also play a major role. Training data for ML techniques could link RTL descriptions to netlists, at the subcircuit level or for entire designs. Another promising application of RTL RE lies in updating legacy designs, as described in [KEP25]. For example, a user might desire to reproduce a legacy design (built at a larger technology node) for a smaller, more advanced technology. If they had access to a gate-level netlist, they could extract the RTL with the proposed method and resynthesize the design in a newer technology.

5 Conclusion

In this study, we presented a comprehensive survey of numerous logical RE techniques, categorized them by their objective and methodology, and analyzed their evolution over time. Furthermore, a detailed description of each RE technique was presented; we compared and contrasted techniques within the same categories, and shed insights into the reappearance and advancements of key RE strategies. Besides discussing their successes, we assessed the drawbacks and areas for improvement of the summarized techniques, with the aim of inspiring future research to address these areas. We also identified and discussed common trends and patterns across the logical RE literature.

Overall, the field of logical RE has made substantial progress across a wide range of objectives. Despite this progress, several challenges remain unresolved, particularly in scalability, evaluation standardization, robustness to obfuscation and noise, extraction of non-functional characteristics, synthesis tool identification, determination of RTL authorship, and full RTL reconstruction. Given the growing importance of IC design in a global and security-sensitive context, continued research is imperative to integrate existing techniques into more practical end-to-end workflows and to address these open problems that have received little or no attention to date.

References

[AHT+20] Nils Albartus, Max Hoffmann, Sebastian Temme, Leonid Azriel, and Christof Paar. DANA - universal dataflow analysis for gate-level netlist reverse engineering. Cryptology ePrint Archive, Paper 2020/751, 2020.

[AK93] C. J. Alpert and A. B. Kahng. Geometric embeddings for faster and better multi-way netlist partitioning. In *Proceedings of the 30th International Design Automation Conference*, DAC '93, page 743–748, New York, NY, USA, 1993. Association for Computing Machinery.

[AK96] C.J. Alpert and A.B. Kahng. Simple eigenvector-based circuit clustering can be effective [vlsi cad]. In *1996 IEEE International Symposium on Circuits and Systems. Circuits and Systems Connecting the World. ISCAS 96*, volume 4, pages 683–686, 1996.

[ASK+21] Lilas Alrahis, Abhrajit Sengupta, Johann Knechtel, Satwik Patnaik, Hani Saleh, Baker Mohammad, Mahmoud Al-Qutayri, and Ozgur Sinanoglu. Gnnre: 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, 2021.

[AV93] S. Areibi and A. Vannelli. A combined eigenvector tabu search approach for circuit partitioning. In *Proceedings of IEEE Custom Integrated Circuits Conference - CICC '93*, pages 9.7.1–9.7.4, 1993.

[BAP+22] Tim Bücher, Lilas Alrahis, Guilherme Paim, Sergio Bampi, Ozgur Sinanoglu, and Hussam Amrouch. Appgnn: Approximation-aware functional reverse engineering using graph neural networks. In *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, New York, NY, USA, 2022. Association for Computing Machinery.

[BBF+08] M. Brutscheck, St. Berger, M. Franke, A. Th. Schwarzbacher, and St. Becker. Structural division procedure for efficient ic analysis. In *IET Irish Signals and Systems Conference (ISSC 2008)*, pages 18–23, 2008.

[BBK89] F. Brglez, D. Bryan, and K. Kozminski. Combinational profiles of sequential benchmark circuits. In *1989 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1929–1934 vol.3, 1989.

[BBS19] Michaela Brunner, Johanna Baehr, and Georg Sigl. Improving on state register identification in sequential hardware reverse engineering. In *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 151–160, 2019.

[BBSS20] Johanna Baehr, Alessandro Bernardini, Georg Sigl, and Ulf Schlichtmann. Machine learning and structural characteristics for reverse engineering. *Integration*, 72:1–12, 2020.

[BF85] Franc Brglez and Hideo Fujiwara. A neutral netlist of 10 combinational benchmark circuits and a targeted translator in fortran. In *1985 IEEE International Symposium on Circuits and Systems*, 06 1985.

[BHB+22] Johanna Baehr, Alexander Hepp, Michaela Brunner, Maja Malenko, and Georg Sigl. Open source hardware design and hardware reverse engineering: A security analysis. In *2022 25th Euromicro Conference on Digital System Design (DSD)*, pages 504–512, 2022.

[BHBS22] Michaela Brunner, Alexander Hepp, Johanna Baehr, and Georg Sigl. Toward a human-readable state machine extraction. *ACM Trans. Des. Autom. Electron. Syst.*, 27(6), June 2022.

[BKW13] Rafal Baranowski, Michael A. Kochte, and Hans-Joachim Wunderlich. Securing access to reconfigurable scan networks. In *2013 22nd Asian Test Symposium*, pages 295–300, 2013.

[BSF+09] M. Brutscheck, B. Schmidt, M. Franke, A. Th. Schwarzbacher, and St. Becker. Identification of deterministic sequential finite state machines in unknown cmos ics. In *IET Irish Signals and Systems Conference (ISSC 2009)*, pages 1–6, 2009.

[BSF+10] M. Brutscheck, B. Schmidt, M. Franke, A. Th. Schwarzbacher, and St. Becker. Optimisation and implementation of a nonlinear identification procedure for unknown ics. In *IET Irish Signals and Systems Conference (ISSC 2010)*, pages 192–197, 2010.

[BSH12] Florian Benz, André Seffrin, and Sorin A. Huss. Bil: A tool-chain for bitstream reverse-engineering. In *22nd International Conference on Field Programmable Logic and Applications (FPL)*, pages 735–738, 2012.

[CBC+23] Animesh B. Chowdhury, Jitendra Bhandari, Luca Collini, Ramesh Karri, Benjamin Tan, and Siddharth Garg. Converts: Contrastively learning structurally invariant netlist representations. In *2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD)*, pages 1–6, 2023.

[CELV99] G.H. Chisholm, S.T. Eckmann, C.M. Lain, and R.L. Veroff. Understanding integrated circuits. *IEEE Design & Test of Computers*, 16(2):26–37, 1999.

[CHCL21] Aijiao Cui, Chengkang He, Chip-Hong Chang, and Hao Lu. Identification of fsm state registers by analytics of scan-dump data. *IEEE Transactions on Information Forensics and Security*, 16:5138–5153, 2021.

[CM18] Burcin Cakir and Sharad Malik. Reverse engineering digital ics through geometric embedding of circuit graphs. *ACM Trans. Des. Autom. Electron. Syst.*, 23(4), July 2018.

[CM19] Burcin Cakir and Sharad Malik. Revealing cluster hierarchy in gate-level ics using block diagrams and cluster estimates of circuit embeddings. *ACM Trans. Des. Autom. Electron. Syst.*, 24(5), June 2019.

[CRSB16] Jacob Couch, Elizabeth Reilly, Morgan Schuyler, and Bradley Barrett. Functional block identification in circuit design recovery. In *2016 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 75–78, 2016.

[CTL01] Wei-Hsin Chang, Shuenn-Der Tzeng, and Chen-Yi Lee. A novel subcircuit extraction algorithm by recursive identification scheme. In *ISCAS 2001. The 2001 IEEE International Symposium on Circuits and Systems (Cat. No.01CH37196)*, volume 5, pages 491–494, 2001.

[CYN21] Subhajit Dutta Chowdhury, Kaixin Yang, and Pierluigi Nuzzo. Reignn: State register identification using graph neural networks for circuit reverse engineering. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9, 2021.

[Dav99] S. Davidson. ITC'99 benchmark circuits - preliminary results. In *International Test Conference 1999. Proceedings (IEEE Cat. No.99CH37034)*, pages 1125–1125, 1999.

[DB17] Yu-Yun Dai and Robert K. Braytont. Circuit recognition with deep learning. In *2017 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 162–162, 2017.

[DCL24] Zhaoxuan Dong, Aijiao Cui, and Hao Lu. Non-invasive reverse engineering of one-hot finite state machines using scan dump data. *IEEE Transactions on Emerging Topics in Computing*, 12(3):795–809, 2024.

[DGP+23] Meng Dong, Yiming Gao, Weitao Pan, Zhiliang Qiu, Jianlei Yang, Zhixiong Di, and Ling Zheng. Relic-gnn: an efficient state register identification algorithm. *Journal of Xidian University*, 50(3):142, 2023.

[Die25] Dietze, Claus and Alphey, Jay and Osorio, Miguel and Klier, Ran. Opentitan. https://opentitan.org/, 2019-2025.

[DWWC98] T. Doom, J. White, A. Wojcik, and G. Chisholm. Identifying high-level components in combinational circuits. In *Proceedings of the 8th Great Lakes Symposium on VLSI (Cat. No.98TB100222)*, pages 313–318, 1998.

[EC97] S. T. Eckmann and G. H. Chisholm. Assigning functional meaning to digital circuits. Technical report, Argonne National Lab. (ANL), Argonne, IL (United States), 07 1997.

[FSN+19] Arash Fayyazi, Soheil Shababi, Pierluigi Nuzzo, Shahin Nazarian, and Massoud Pedram. Deep learning-based circuit recognition using sparse mapping and level-dependent decaying sum circuit representations. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 638–641, 2019.

[FWD+19] Marc Fyrbiak, Sebastian Wallat, Jonathan Déchelotte, Nils Albartus, Sinan Böcker, Russell Tessier, and Christof Paar. On the difficulty of FSM-based hardware obfuscation. Cryptology ePrint Archive, Paper 2019/1163, 2019.

[FWR+19] Marc Fyrbiak, Sebastian Wallat, Sascha Reinhard, Nicolai Bissantz, and Christof Paar. Graph similarity and its applications to hardware security. *IEEE Transactions on Computers*, 69(4):505–519, 2019.

[FWS+19] 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.

[GASA24a] Paul R. Genssler, Lilas Alrahis, Ozgur Sinanoglu, and Hussam Amrouch. Circuithd: Brain-inspired hyperdimensional computing for circuit recognition. *IEEE Access*, 12:176280–176290, 2024.

[GASA24b] Paul R. Genssler, Lilas Alrahis, Ozgur Sinanoglu, and Hussam Amrouch. Hdcircuit: Brain-inspired hyperdimensional computing for circuit recognition. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–2, 2024.

[GMZJ20] James Geist, Travis Meade, Shaojie Zhang, and Yier Jin. Relic-fun: Logic identification through functional signal comparisons. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2020.

[GMZJ23a] James Geist, Travis Meade, Shaojie Zhang, and Yier Jin. Netviz: A tool for netlist security visualization. In *2023 24th International Symposium on Quality Electronic Design (ISQED)*, pages 1–8, 2023.

[GMZJ23b] Jim Geist, Travis Meade, Shaojie Zhang, and Yier Jin. Improving fsm state enumeration performance for hardware security with recut and refsm-sat, 2023.

[Gre13] Michael Green. Digital netlist partitioning system for faster circuit reverse-engineering, 2013. U.S. Patent 8 413 085.

[GSD+14] Adrià Gascón, Pramod Subramanyan, Bruno Dutertre, Ashish Tiwari, Dejan Jovanović, and Sharad Malik. Template-based circuit understanding. In *2014 Formal Methods in Computer-Aided Design (FMCAD)*, pages 83–90, 2014.

[HBS22] Alexander Hepp, Johanna Baehr, and Georg Sigl. Golden model-free hardware trojan detection by classification of netlist module graphs. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1317–1322, 2022.

[HCC19] Chengkang He, Aijiao Cui, and Chip-Hong Chang. Identification of state registers of fsm through full scan by data analytics. In *2019 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, pages 1–6, 2019.

[HK92] L. Hagen and A. Kahng. A new approach to effective circuit clustering. In *1992 IEEE/ACM International Conference on Computer-Aided Design*, pages 422–427, 1992.

[HLSG21] Xuenong Hong, Tong Lin, Yiqiong Shi, and Bah Hwee Gwee. Asic circuit netlist recognition using graph neural network. In *2021 IEEE International Symposium on the Physical and Failure Analysis of Integrated Circuits (IPFA)*, pages 1–5, 2021.

[HMV92] S.W. Hadley, B.L. Mark, and A. Vannelli. An efficient eigenvector approach for finding netlist partitions. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(7):885–892, 1992.

[HWB+21] Zhuolun He, Ziyi Wang, Chen Bai, Haoyu Yang, and Bei Yu. Graph learning-based arithmetic block identification. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–8, 2021.

[HYH99] M.C. Hansen, H. Yalcin, and J.P. Hayes. Unveiling the iscas-85 benchmarks: a case study in reverse engineering. *IEEE Design & Test of Computers*, 16(3):72–80, 1999.

[KEP25] Adam Kimura, Andrew Elliott, and Daniel Perkins. Recovery of a hierarchical functional representation of an integrated circuit, 2025. U.S. Patent 0 165 686.

[KFFT22] Rasheed Kibria, Nusrat Farzana, Farimah Farahmandi, and Mark Tehranipoor. Fsmx: Finite state machine extraction from flattened netlist with application to security. In *2022 IEEE 40th VLSI Test Symposium (VTS)*, pages 1–7, 2022.

[KFT23] Rasheed Kibria, Farimah Farahmandi, and Mark Tehranipoor. Fsmx-ultra: Finite state machine extraction from gate-level netlist for security assessment. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 42(11):3613–3627, 2023.

[LGS+13] Wenchao Li, Adria Gascon, Pramod Subramanyan, Wei Yang Tan, Ashish Tiwari, Sharad Malik, Natarajan Shankar, and Sanjit A. Seshia. Wordrev: Finding word-level structures in a sea of bit-level gates. In *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 67–74, 2013.

[LKK+20] Nimisha Limaye, Emmanouil Kalligeros, Nikolaos Karousos, Irene G. Karybali, and Ozgur Sinanoglu. Thwarting all logic locking attacks: Dishonest oracle with truly random logic locking. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(9):1740–1753, 2020.

[LKS+22] Min Li, Sadaf Khan, Zhengyuan Shi, Naixing Wang, Huang Yu, and Qiang Xu. Deepgate: learning neural representations of logic gates. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, page 667–672, New York, NY, USA, 2022. Association for Computing Machinery.

[LT79] Thomas Lengauer and Robert Endre Tarjan. A fast algorithm for finding dominators in a flowgraph. *ACM Trans. Program. Lang. Syst.*, 1(1):121–141, January 1979.

[LWS12] Wenchao Li, Zach Wasson, and Sanjit A. Seshia. Reverse engineering circuits using behavioral pattern mining. In *2012 IEEE International Symposium on Hardware-Oriented Security and Trust*, pages 83–88, 2012.

[LYL25] Zongtai Li, Liang Yang, and Mian Lou. Dagnn-re: Directed acyclic graph neural network for functional reverse engineering of gate-level netlist. *Integration*, 102:1–18, 2025.

[LYZH16] Duo Liu, Cunxi Yu, Xiangyu Zhang, and Daniel Holcomb. Oracle-guided incremental sat solving to reverse engineer camouflaged logic circuits. In *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 433–438, 2016.

[MBG+24] Michael Mildner, Michaela Brunner, Michael Gruber, Johanna Baehr, and Georg Sigl. Fault-simulation-based flip-flop classification for reverse engineering. In *2024 27th International Symposium on Design & Diagnostics of Electronic Circuits & Systems (DDECS)*, pages 53–56, 2024.

[McE01] Kenneth McElvain. Methods and apparatuses for automatic extraction of finite state machines, 2001. U.S. Patent 6 182 268.

[MGT17] Mohamed El Massad, Siddharth Garg, and Mahesh Tripunitara. Reverse engineering camouflaged sequential circuits without scan access. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 33–40, 2017.

[MJTZ16] Travis Meade, Yier Jin, Mark Tehranipoor, and Shaojie Zhang. Gate-level netlist reverse engineering for hardware security: Control logic register identification. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1334–1337, 2016.

[MPAR22] Jeffrey Mcdonald, Jennifer Parnell, Todd Andel, and Samuel Russ. Effectiveness of adversarial component recovery in protected netlist circuit designs. In *Proceedings of the 19th International Conference on Security and Cryptograph*, pages 181–192, 01 2022.

[MPZJ19] Travis Meade, Jason Portillo, Shaojie Zhang, and Yier Jin. Neta: when ip fails, secrets leak. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, page 90–95, New York, NY, USA, 2019. Association for Computing Machinery.

[MSL+18] Travis Meade, Kaveh Shamsi, Thao Le, Jia Di, Shaojie ZHang, and Yier Jin. The old frontier of reverse engineering: Netlist partitioning. *Journal of Hardware and Systems Security*, 2:201–213, 2018.

[MSNG22] Reilly McKendrick, Corey Simpson, Brent Nelson, and Jeffrey Goeders. Leveraging fpga primitives to improve word reconstruction during netlist reverse engineering. In *2022 International Conference on Field-Programmable Technology (ICFPT)*, pages 1–5, 2022.

[MZJ16a] Travis Meade, Shaojie Zhang, and Yier Jin. Netlist reverse engineering for high-level functionality reconstruction. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 655–660, 2016.

[MZJ+16b] Travis Meade, Shaojie Zhang, Yier Jin, Zheng Zhao, and David Pan. Gate-level netlist reverse engineering tool set for functionality recovery and malicious logic detection. In *ISTFA 2016: Conference Proceedings from the 42nd International Symposium for Testing and Failure Analysis*, International Symposium for Testing and Failure Analysis, pages 342–346, 11 2016.

[NSS94] Esko Nuutila and Eljas Soisalon-Soininen. On finding the strongly connected components in a directed graph. *Information Processing Letters*, 49(1):9–14, 1994.

[NVP+23] Ram Venkat Narayanan, Aparajithan Nathamuni Venkatesan, Kishore Pula, Sundarakumar Muthukumaran, and Ranga Vemuri. Reverse engineering word-level models from look-up table netlists. In *2023 24th International Symposium on Quality Electronic Design (ISQED)*, pages 1–8, 2023.

[NXY+16] Adib Nahiyan, Kan Xiao, Kun Yang, Yeir Jin, Domenic Forte, and Mark Tehranipoor. Avfsm: a framework for identifying and mitigating vulnerabilities in fsms. In *Proceedings of the 53rd Annual Design Automation Conference*, New York, NY, USA, 2016. Association for Computing Machinery.

[OEGS93] M. Ohlrich, C. Ebeling, E. Ginting, and L. Sather. Subgemini: Identifying subcircuits using a fast subgraph isomorphism algorithm. In *30th ACM/IEEE Design Automation Conference*, pages 31–37, 1993.

[Oli25] Oliscience BV. Opencores: Community for open-source digital designs. https://opencores.org/, 2019-2025.

[PAISR24] Samuel Pagliarini, Aikata Aikata, Malik Imran, and Sujoy Sinha Roy. Repqc: Reverse engineering and backdooring hardware accelerators for post-quantum cryptography. In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, page 533–547, New York, NY, USA, 2024. Association for Computing Machinery.

[PBN94] I. Parulkar, M.A. Breuer, and C.A. Njinda. Extraction of a high level structural representation from circuit descriptions with applications to dft/bist "lambda. In *31st Design Automation Conference*, pages 345–350, 1994.

[PCS+21] Christian Pilato, Animesh Basak Chowdhury, Donatella Sciuto, Siddharth Garg, and Ramesh Karri. Assure: Rtl locking against an untrusted foundry. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 29(7):1306–1318, 2021.

[PMH+19] Jason Portillo, Travis Meade, John Hacker, Shaojie Zhang, and Yier Jin. Rertl: Finite state transducer logic recovery at register transfer level. *2019 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*, 2019.

[PR94] G. Pelz and U. Roettcher. Pattern matching and refinement hybrid approach to circuit comparison. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(2):264–276, 1994.

[RDJ94] B.M. Riess, K. Doll, and F.M. Johannes. Partitioning very large circuits using analytical placement techniques. In *31st Design Automation Conference*, pages 646–651, 1994.

[RKK14] Masoud Rostami, Farinaz Koushanfar, and Ramesh Karri. A primer on hardware security: Models, methods, and metrics. *Proceedings of the IEEE*, 102(8):1283–1295, 2014.

[RR22] Sreeja Rajendran and Mary Lourde Regeena. A novel algorithm for hardware trojan detection through reverse engineering. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 41(4):1154–1166, 2022.

[RS24] Varun Rao and Zachary D. Sisco. Register aggregation for hardware decompilation. *arXiv*, 2024. https://doi.org/10.48550/arXiv.2409.03119.

[Rub03] N. Rubanov. Subislands: the probabilistic match assignment algorithm for subcircuit recognition. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(1):26–38, 2003.

[Sar25] Alexander Sarti. Beyond the chips act: Investing in a technological leap. *Yale Journal of International Affairs*, 2025.

[SBSH23] Zachary D. Sisco, Jonathan Balkind, Timothy Sherwood, and Ben Hardekopf. Loop rerolling for hardware decompilation. In *Proceedings of the ACM on Programming Languages*, volume 7, New York, NY, USA, June 2023. Association for Computing Machinery.

[SGR+12] Yiqiong Shi, Bah-Hwee Gwee, Ye Ren, Thet Khaing Phone, and Chan Wai Ting. Extracting functional modules from flattened gate-level netlist. In *2012 International Symposium on Communications and Information Technologies (ISCIT)*, pages 538–543, 2012.

[SIA18] Detecting and removing counterfeit semiconductors in the U.S. supply chain. Technical report, Semiconductor Industry Association, 2018.

[SLB05] Blair Schiffner, Jianhua Li, and Laleh Behjat. A Multivalue Eigenvalue Based Circuit Partitioning Technique . In *Proceedings. Fifth International Workshop on System-on-Chip for Real-Time Applications*, pages 312–316, Los Alamitos, CA, USA, July 2005. IEEE Computer Society.

[Smi13] Jessica Smith. Non-destructive state machine reverse engineering. In *2013 6th International Symposium on Resilient Control Systems (ISRCS)*, pages 120–124, 2013.

[SOMM17] Jessica Smith, Kiri Oler, Carl Miller, and David Manz. Reverse engineering integrated circuits using finite state machine analysis. In *Hawaii International Conference on System Sciences 2017 (HICSS-50)*, pages 2906–2914, 2017.

[SPK+23] Zhengyuan Shi, Hongyang Pan, Sadaf Khan, Min Li, Yi Liu, Junhua Huang, Hui-Ling Zhen, Mingxuan Yuan, Zhufei Chu, and Qiang Xu. Deepgate2: Functionality-aware circuit representation learning. In *2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 1–9, 2023.

[SSDB15] Mathias Soeken, Baruch Sterin, Rolf Drechsler, and Robert Brayton. Simulation graphs for reverse engineering. In *2015 Formal Methods in Computer-Aided Design (FMCAD)*, pages 152–159, 2015.

[STGR10] Yiqiong Shi, Chan Wai Ting, Bah-Hwee Gwee, and Ye Ren. A highly efficient method for extracting fsms from flattened gate-level netlist. In *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*, pages 2610–2613, 2010.

[STL+14] Pramod Subramanyan, Nestan Tsiskaridze, Wenchao Li, Adrià Gascón, Wei Yang Tan, Ashish Tiwari, Natarajan Shankar, Sanjit A. Seshia, and Sharad Malik. Reverse engineering digital circuits using structural and functional analyses. *IEEE Transactions on Emerging Topics in Computing*, 2(1):63–80, 2014.

[STP+13] Pramod Subramanyan, Nestan Tsiskaridze, Kanika Pasricha, Dillon Reisman, Adriana Susnea, and Sharad Malik. Reverse engineering digital circuits using functional analysis. In *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1277–1280, 2013.

[Str25] Joachim Strömbergson. Repositories of open-source verilog designs. https://github.com/secworks, 2012-2025.

[Tar71] Robert Tarjan. Depth-first search and linear graph algorithms. In *12th Annual Symposium on Switching and Automata Theory (swat 1971)*, pages 114–121, 1971.

[TD15] Edward Tashjian and Azadeh Davoodi. On using control signals for word-level identification in a gate-level netlist. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2015.

[TJ09] Randy Torrance and Dick James. The state-of-the-art in ic reverse engineering. In Christophe Clavier and Kris Gaj, editors, *Cryptographic Hardware and Embedded Systems - CHES 2009*, pages 363–381, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[TLH+24] Wuqian Tang, Yi-Ting Li, Kai-Po Hsu, Kuan-Ling Chou, You-Cheng Lin, Chia-Feng Chien, Tzu-Li Hsu, Yung-Chih Chen, Ting-Chi Wang, Shih-Chieh Chang, TingTing Hwang, and Chun-Yao Wang. A hybrid approach to reverse engineering on combinational circuits. In *2024 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–2, 2024.

[Vam18] Harsh Vamja. Reverse engineering of finite state machines from sequential circuits. Master's thesis, University of Cincinnati, 2018.

[VAV19] Harsh Vamja, Richa Agrawal, and Ranga Vemuri. Non-invasive reverse engineering of finite state machines using power analysis and boolean satisfiability. In *2019 IEEE 62nd International Midwest Symposium on Circuits and Systems (MWSCAS)*, pages 452–455, 2019.

[VR96] N. Vijaykrishnan and N. Ranganathan. Subgen: a genetic approach for subcircuit extraction. In *Proceedings of 9th International Conference on VLSI Design*, pages 343–345, 1996.

[WAB+19] Sebastian Wallat, Nils Albartus, Steffen Becker, Max Hoffmann, Maik Ender, Marc Fyrbiak, Adrian Drees, Sebastian Maaßen, and Christof Paar. Highway to hal: open-sourcing the first extendable gate-level netlist reverse engineering framework. In *Proceedings of the 16th ACM International Conference on Computing Frontiers*, page 392–397. Association for Computing Machinery, 2019.

[WBH+22] Ziyi Wang, Chen Bai, Zhuolun He, Guangliang Zhang, Qiang Xu, Tsung-Yi Ho, Bei Yu, and Yu Huang. Functionality matters in netlist representation learning. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*, page 61–66, New York, NY, USA, 2022. Association for Computing Machinery.

[WBHS22] Selina Weber, Johanna Baehr, Alexander Hepp, and Georg Sigl. Analysis of Graph-based Partitioning Algorithms and Partitioning Metrics for Hardware Reverse Engineering. In *11th International Workshop on Security Proofs for Embedded Systems (PROOFS 2022)*, Leuven, Belgium, September 2022.

[WDLW15] Xing Wei, Yi Diao, Tak-Kei Lam, and Yu-Liang Wu. A universal macro block mapping scheme for arithmetic circuits. In *2015 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1629–1634, 2015.

[WLBG18] Michael Werner, Bernhard Lippmann, Johanna Baehr, and Helmut Gräb. Reverse engineering of cryptographic cores by structural interpretation through graph analysis. In *2018 IEEE 3rd International Verification and Security Workshop (IVSW)*, pages 13–18, 2018.

[WLH+23] Nan Wu, Yingjie Li, Cong Hao, Steve Dai, Cunxi Yu, and Yuan Xie. Gamora: Graph learning based symbolic reasoning for large-scale boolean networks. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, pages 1–6, 2023.

[WWCD00] Jennifer L. White, Anthony S. Wojcik, Moon-Jung Chung, and Travis E. Doom. Candidate subcircuits for functional module identification in logic circuits. In *Proceedings of the 10th Great Lakes Symposium on VLSI*, GLSVLSI '00, page 34–38, New York, NY, USA, 2000. Association for Computing Machinery.

[YC16] Cunxi Yu and Maciej Ciesielski. Automatic word-level abstraction of datapath. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 1718–1721, 2016.

[YS03] Lei Yang and C.-J.R. Shi. Frosty: a fast hierarchy extractor for industrial cmos circuits. In *ICCAD-2003. International Conference on Computer Aided Design (IEEE Cat. No.03CH37486)*, pages 741–746, 2003.

[YYNF21] Rozhin Yasaei, Shih-Yuan Yu, Emad Kasaeyan Naeini, and Mohammad Abdullah Al Faruque. Gnn4ip: Graph neural network for hardware intellectual property piracy detection. In *2021 58th ACM/IEEE Design Automation Conference (DAC)*, pages 217–222, 2021.

[ZDT25] Lizi Zhang, Azadeh Davoodi, and Rasit Onur Topaloglu. Rebert: Llm for gate-level to word-level reverse engineering. In *2025 Design, Automation & Test in Europe Conference (DATE)*, pages 1–7, 2025.

[ZS22] Guangwei Zhao and Kaveh Shamsi. Graph neural network based netlist operator detection under circuit rewriting. In *Proceedings of the Great Lakes Symposium on VLSI 2022*, page 53–58, New York, NY, USA, 2022. Association for Computing Machinery.

[ZW03] Nian Zhang and D.C. Wunsch. A fuzzy attributed graph approach to subcircuit extraction problem. In *The 12th IEEE International Conference on Fuzzy Systems, 2003. FUZZ '03.*, volume 2, pages 1063–1067 vol.2, 2003.