J Cryptol (2026) 39:6
https://doi.org/10.1007/s00145-025-09563-4

Journal of CRYPTOLOGY

Research Article

Faster Homomorphic Operations and Beyond: Expediting Homomorphic Computation via Boolean Circuit Optimization

Mingfei Yu · Giovanni De Micheli
EPFL, Lausanne, Switzerland
mingfei.yu@epfl.ch
giovanni.demicheli@epfl.ch

Communicated by Gregor Leander and Svetla Nikova

Received 23 September 2024 / Revised 13 October 2025 / Accepted 22 October 2025

Abstract. Fully homomorphic encryption (FHE) enables secure data processing without compromising data access. However, its computational cost and slower execution compared to plaintext operations present significant challenges. The increasing interest in FHE-based secure computation underscores the need to accelerate homomorphic computations. Existing research predominantly focuses on reducing the multiplicative depth (MD) of FHE circuits, as a lower MD enhances the execution efficiency of each homomorphic operation. However, this often comes at the expense of increased multiplicative complexity (MC), leading to more homomorphic multiplications — a computationally intensive task. Currently, there is a lack of approaches that effectively balance the trade-off between MD reduction and MC increase, potentially resulting in suboptimal outcomes. This paper addresses this critical gap with three main contributions: (a) an exact synthesis paradigm for generating optimal FHE circuit implementations, (b) a heuristic circuit optimization algorithm, named MC-aware MD minimization, that leverages the exact synthesis paradigm to optimize FHE circuits efficiently, and (c) an FHE circuit optimization flow that integrates MC-aware MD minimization with existing MD reduction techniques. Experimental results demonstrate a 21.32% average reduction in homomorphic computation time and highlight significantly improved efficiency in circuit optimization.

Keywords. Homomorphic Encryption, Boolean Circuits, Logic Synthesis, Multiplicative Depth, Multiplicative Complexity.

1. Introduction

First recognized in 1978, the concept of *fully homomorphic encryption* (FHE) refers to a revolutionary encryption paradigm that enables direct computation on ciphertexts without the need for decryption [33]. The groundbreaking *bootstrapping* theorem by Gentry marks the birth of the first HE scheme capable of supporting arbitrary computa-

Published online: 12 November 2025

tions, known as *leveled FHE* [21]. Whereas continuous developments have significantly improved the practicality of FHE, homomorphic computation is typically orders of magnitude slower than its counterpart on plaintexts. Despite the computational challenges, FHE offers the unique advantage of delegating data processing without compromising data access. This characteristic positions FHE as a promising solution for securing computations, with numerous potential applications in scenarios where privacy is paramount. Examples include outsourcing medical data for diagnosis [14] and privacy-preserving neural network inference [18]. The growing interest in FHE-based secure computation techniques underscores the need for continuous efforts to accelerate homomorphic computations.

Modern FHE schemes broadly fall into two categories: *leveled schemes*, such as *BGV* [4], *BFV* [20], and *CKKS* [12], and *fast-bootstrapping schemes*, exemplified by *FHEW* [17] and *TFHE* [11]. Although a more detailed discussion is deferred to the following section, we emphasize that these two families are based on fundamentally different constructions and therefore present distinct considerations for performance optimization. In this work, we constrain our discussion to leveled schemes, and specifically focus on the logic-level optimization of Boolean circuits to accelerate their homomorphic evaluation under schemes like BFV and BGV, which support exact computation over encrypted bits.

Leveled schemes operate over large plaintext moduli and use techniques such as modulus switching to manage noise growth. With a sufficient noise budget — i.e., a sufficiently large ciphertext modulus composed of multiple prime factors — a circuit can be evaluated homomorphically without requiring intermediate bootstrapping [31]. A homomorphic encryption scheme is considered functionally complete if it supports both additions and multiplications. Homomorphic multiplication is significantly more expensive than addition for two primary reasons: (a) it incurs substantially higher computational complexity; and (b) each homomorphic multiplication results in quadratic noise accumulation and consumes one level of the noise budget. Consequently, a circuit with a longer chain of multiplications requires a larger noise budget (i.e., more levels) to accommodate the accumulated noise. This implies that, in leveled FHE schemes, the execution time of each homomorphic operation is closely related to the *multiplicative depth* (MD) of the circuit implementing the target computation. The MD of a circuit is defined as the maximum number of sequential homomorphic multiplications along any path. Evaluating an FHE circuit with high MD either necessitates the use of very large encryption parameters or incurs the overhead of expensive bootstrapping — both of which significantly increase runtime. As a result, a central challenge in accelerating homomorphic computation is to develop circuit implementations with minimal MD for the given functionality.

Reducing the MD of Boolean circuits has attracted significant research attention [1,8,9,28]. This interest is largely driven by the natural compatibility between the binary plaintext space and high-level programming language instructions, which enables seamless compilation of conventional software programs into homomorphically encrypted counterparts for secure execution [9]. In this context, the plaintext space is binary, and addition and multiplication correspond to Boolean XOR and AND operations, respectively. As a result, target computations can be modeled as Boolean functions, allowing the application of logic synthesis techniques to restructure circuits for reduced MD.

However, minimizing circuit MD commonly comes at the cost of increasing the total number of multiplications, known as the circuit s *multiplicative complexity* (MC). This is because optimization techniques that aggressively reduce MD may introduce many parallel multiplications (e.g., trading two chained multiplications for several parallel ones), thereby increasing the total count of homomorphic multiplications. The challenge is that MD and MC jointly determine the overall homomorphic execution time. Existing circuit optimization tools have largely treated the MD MC trade-off in an ad hoc way — for example, by halting MD reduction passes once a certain MC threshold is exceeded — rather than optimizing both criteria together. This gap motivates a more holistic approach to FHE circuit optimization.

In this paper, we present three key contributions to address this challenge. Terming the cost function of an FHE circuit, which depends on both the MC and MD of the circuit, as *FHE cost*, we propose: (a) an exact logic synthesis framework for finding FHE-cost-optimal circuit realizations for small Boolean functions, exploring the optimal trade-offs between MD and MC (Section 3); (b) a scalable *MC-aware MD minimization* algorithm that uses the exact synthesis results as a subroutine to optimize larger circuits FHE cost (Section 4); and (c) an integrated FHE circuit optimization flow that combines our MC-aware optimization with state-of-the-art MD-only reduction techniques from the literature (specifically, we incorporate the *ESOP balancing* method [26]). While the optimal formulation of FHE cost remains an open question, we adopt the cost metric MC $\times$ MD^2, following empirical observations reported in prior work [15]. Using this configuration, our experimental results demonstrate that the proposed approach achieves substantial speedups in the homomorphic evaluation of Boolean functions compared to prior MD-optimized circuits — yielding an average improvement of 21.32%, along with a four-orders-of-magnitude reduction in circuit optimization time. These results mark a meaningful step toward narrowing the performance gap between encrypted and plaintext computations.

2. Background

This section provides preliminaries on mainstream FHE schemes (Section 2.1) and Boolean circuit notations (Section 2.2), a summary of existing research on homomorphic computation acceleration via FHE circuits operation (Section 2.3.1 to Section 2.3.3), and the motivation of developing MC-aware MD minimization technique (Section 2.3.4).

2.1. *FHE Schemes: Leveled vs. Fast Bootstrapping*

FHE schemes can be broadly categorized by how they manage noise growth during homomorphic multiplications. Leveled FHE schemes — including *BGV* [4], *BFV* [20], and *CKKS* [12] — allow a ciphertext to undergo a predetermined number of multiplications (the *multiplicative depth* (MD)) without the need to refresh the embedded noise component. This is achieved by choosing sufficiently large encryption parameters, such as the modulus size and polynomial degree, to provide an adequate noise budget for the required MD. In practice, supporting a circuit of MD L requires $L+1$ modulus “layers” in the ciphertext [31], as each homomorphic multiplication consumes one layer. Once all layers are depleted, further multiplications are only possible via bootstrapping, which is computationally expensive and therefore avoided whenever possible. The principal advantage of leveled schemes is that, for computations within the MD budget, bootstrapping can be entirely bypassed, leading to significantly reduced runtime compared to schemes that rely on frequent bootstrapping. The drawback, however, is that increasing the MD necessitates larger parameters — resulting in greater ciphertext and key sizes, and more costly polynomial arithmetic for each operation. This inherent trade-off underscores the importance of circuit optimization in leveled FHE: by reducing MD, one can select smaller parameters and achieve faster homomorphic operations.

Fast bootstrapping schemes, such as *FHEW* [17] and *TFHE* [11], take a different approach by employing frequent bootstrapping to manage noise growth. These schemes typically operate on ciphertexts encrypting small integers or individual bits and refresh ciphertexts continuously, so that noise remains controlled regardless of circuit MD. In TFHE, for example, every nonlinear gate (such as AND) is computed via a bootstrapping operation that both evaluates the gate and resets the noise, allowing its output to be immediately reused in subsequent operations. This distinguishing feature of TFHE is its *programmable bootstrapping* (PBS) [11], which enables each bootstrap to act as a lookup-table evaluation — thus supporting the computation of arbitrary functions over the encrypted input during refresh. This confers significant flexibility at the logic level, as complex operations can be implemented as a single bootstrapping, provided they can be represented as a truth table. The cost per bootstrapping in such schemes, compared to leveled schemes, is relatively modest — on the order of 5 to 20 milliseconds for a single gate — and remains essentially constant regardless of the circuit s MD [7]. Therefore, unlike leveled schemes, fast-bootstrapping schemes do not require parameter selection based on MD. Instead, the main performance bottleneck is the total number of gates (i.e., PBS operations), not the MD itself.

This fundamental distinction leads to different circuit optimization strategies for the two scheme families. Leveled FHE circuit optimization primarily aims to reduce MD (sometimes at the cost of increased gate count), whereas fast bootstrapping schemes focus on minimizing the total number of bootstrapping operations and leveraging larger, functionally rich gates to amortize bootstrapping costs [7,23,41]. Crucially, optimization techniques developed for fast-bootstrapping schemes are not directly applicable to leveled schemes due to these essential differences. In summary, while fast-bootstrapping FHE prioritizes minimizing the number of bootstraps, leveled FHE circuit optimization centers on reducing MD — even if this results in a moderate increase in *multiplicative complexity* (MC), as targeted in prior works [1,8,9,28]. This paper is devoted to the latter paradigm, addressing the unique performance bottlenecks present in leveled FHE schemes.

2.2. *Boolean circuit*

A Boolean circuit can be represented as a logic network structured as a directed acyclic graph with a node set V and a directed edge set E. Set V comprises three distinct subsets: a set I of *primary inputs* (PIs) lacking fan-in, a set O of *primary outputs* (POs) with singular fan-in and lacking fanout, and a set G of logic gates chosen from a library.

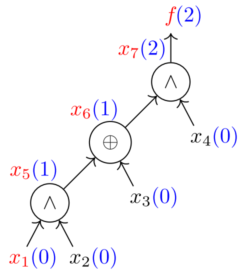

Fig. 1. An XAG implementation of function #7800. The multiplicative depth of each node is displayed in blue, with a critical path highlighted in red.

In FHE schemes configured with a binary plaintext space, multiplications and additions correspond to Boolean AND and XOR operations, respectively. As a result, an FHE circuit is represented by a logic network featuring a gate library of 2-input AND and 2-input XOR gates, with optional input negation, commonly referred to as an *XOR-AND graph* (XAG). Due to this correspondence, we use the terms *network* and *circuit* interchangeably.

An XAG implementing a 4-variable function, whose truth table is #7800[1], is depicted in Fig. 1, where '$\wedge$' and '$\oplus$' denote AND gate and XOR gate, respectively.[2] It consists of PIs $I = \{x_1, x_2, x_3, x_4\}$, POs $O = \{f\}$, gates $G = \{x_5, x_6, x_7\}$, and a set of directed edges E connecting the nodes.

Cut is a concept commonly employed to identify a sub-network or circuit. A cut is characterized by its *root* (a node) and *leaves* (a collection of nodes). A valid set of leaves must satisfy two properties: (1) There is at least one leaf on any path from a PI to the root; (2) All leaves are on at least one such path. A cut is deemed *k-feasible* if its number of leaves does not exceed k, referred to as a k-cut. For example, in Fig. 1, there are two 3-cuts rooted at x_7 with leaves $\{x_3, x_4, x_5\}$ and $\{x_4, x_6\}$, respectively. The process of finding all k-cuts in the target network is known as *cut enumeration* [30].

An XAG with $|G|$ gates can be alternatively modeled as a sequence of $|G|$ *steps*, where each step can be represented as

$$x_i = x_{j_1} \circ_i x_{j_2}, \text{ where } \circ_i \in \{\wedge, \oplus\},$$

for $|I| < i \leq |I| + |G|$ and $1 \leq j_1 < j_2 < i$. If an XAG has a single PO, i.e., $|O| = 1$, the function is computed by the last step $x_{|I|+|G|}$. If a node is a PI, its *multiplicative depth* (MD) is defined to be 0; Otherwise, the MD of step x_i, denoted as σ_i, is recursively defined as

$$\sigma_i = \begin{cases} \max\{\sigma_{j_1}, \sigma_{j_2}\} & \text{if } \circ_i = \oplus \\ \max\{\sigma_{j_1}, \sigma_{j_2}\} + 1 & \text{if } \circ_i = \wedge \end{cases} \tag{1}$$

A PO node's MD matches that of its fan-in. The MD of an XAG, denoted as d, is defined as the maximal MD of its steps (or gates) [8], i.e., $d = \max\{\sigma_i \mid |I| < i \leq |I| + |G|\}$.

[2]In this paper, truth tables are represented in hexadecimal as a bit-string, and the most significant bit is on the left-hand side.

A path, from a PI to a PO, that determines the MD of an XAG is called the *critical path*, such as the one marked in Fig. 1. The *multiplicative complexity* (MC) of an XAG, denoted as c, is defined as the number of AND gates within it [6], i.e., $c = |\{i \mid \circ_i = \wedge, |I| < i \leq |I| + |G|\}|$.

2.3. *Circuit Optimization in Leveled FHE Schemes*

To enable computation on encrypted data, the target function must first be represented as a Boolean or arithmetic circuit compatible with the underlying FHE scheme. In this subsection, we survey the range of research efforts aimed at improving the efficiency of leveled FHE schemes through circuit optimization. We pay particular attention to advances in Boolean circuit optimization, as this is the primary focus of the present work, while also highlighting relevant developments in arithmetic circuit optimization.

2.3.1. *MD Reduction*

In the early stage of Boolean FHE circuit optimization, researchers leveraged circuit depth optimization algorithms within the open-source hardware synthesis tool, `ABC` [5], with the hope that reducing the depth would lead to a decrease in MD [9]. The movement toward customizing optimization algorithms for MD reduction gained momentum with Carpov *et al.*'s seminal work [8]. This work introduced algebraic rules for structurally transforming circuits. For example, applying the associativity of the AND operation, $(a \wedge b) \wedge c = a \wedge (b \wedge c)$, to gate a on a critical path could potentially reduce MD by one unit. The state-of-the-art homomorphic circuit optimization tool, `LOBSTER` [28], employs two key steps for MD reduction: (1) offline rule learning, where a set of MD reduction rules is extracted from a training set of circuits, and (2) online term rewriting, where the learned rules are maximally applied to the target circuit to minimize its MD. While `LOBSTER` surpasses previous techniques as it inherently leverages Boolean properties during the learning stage, it does have the drawback of requiring a prohibitively long time for learning and inefficient pattern matching for maximally applying the learned rules.

Although not widely recognized in the cryptographic literature, *ESOP balancing* [26] stands out as the most performant MD reduction technique. It was initially proposed for *T-depth minimization*, a key aspect of *fault-tolerant quantum computing*. Leveraging *exclusive sum-of-products* (ESOPs)' potential as a low-MD XAG, a circuit's MD can be reduced by balancing the AND trees within the ESOP representation of each sub-circuit (i.e., each cut). Specifically, the algorithm translates each product term (called *cube*) of the ESOP representation into a balanced tree of 2-input AND gates, with the outputs of these AND-trees combined using 2-input XOR gates that do not contribute to the overall MD of the resulting XAG. Note that ESOP balancing leverages Boolean-specific properties and is therefore applicable only to Boolean circuits.

2.3.2. *Arithmetic Circuit vs. Boolean Circuit*

An orthogonal line of research in FHE circuit optimization focuses on higher-level, datapath-oriented optimizations, rather than bit-level logic. Many FHE applications exploit the fact that most leveled schemes (such as BGV and BFV) support arithmetic over large plaintext moduli, or — when using CKKS — even approximate arithmetic over real numbers. By leveraging this feature, it is possible to perform more computation per homomorphic operation. For example, rather than implementing 32-bit integer addition as 32 binary adders, one can instead perform a single ciphertext addition, provided the plaintext modulus is sufficiently large to avoid overflow. Recent studies demonstrate that significant speedups can be achieved by selecting suitable high-level algorithms or employing arithmetic circuits that are more efficient in the homomorphic setting [22,39].

Importantly, these high-level approaches are not in conflict with Boolean (bit-level) circuit optimizations, which are the focus of this work. On the contrary, they are complementary. A typical workflow may first apply high-level optimizations, such as using native integer operations, to reduce the complexity or depth of a computation. Subsequently, low-level Boolean circuit optimizations can be applied to those components that must still be evaluated at the bit level.

This paper is primarily concerned with the latter: logic-level optimization of Boolean circuits for leveled FHE. This focus is orthogonal to the algorithmic or word-level optimizations implemented in advanced FHE circuit synthesizers, commonly referred to as *FHE compilers*. In principle, an FHE compiler could integrate our Boolean circuit optimization techniques as a backend pass, following high-level transformations. By addressing optimization at the gate level, our work enhances the final stage of FHE program optimization and can be effectively combined with more abstract, upstream optimizations.

In summary, accelerating homomorphic computation is a multifaceted challenge that spans scheme selection, high-level algorithmic design, and low-level gate optimization. The techniques presented in this paper represent an advance in the low-level (logic synthesis) domain and are designed to dovetail with improvements in other layers of the FHE toolchain. Furthermore, the discussion section will explore how our methodology for Boolean FHE circuit synthesis could be adapted to yield improved arithmetic circuit designs.

2.3.3. *Automatic Bootstrapping Management*

In certain application scenarios, such as *private neural network inference*, where the MD of the corresponding FHE circuits tend to be large, applying the bootstrapping operation to refresh the noise level of ciphertexts can lead to more efficient homomorphic computation [13]. The challenge of determining the optimal moment to apply bootstrapping is known as the *automatic bootstrapping management* problem. Although coupling automatic bootstrapping management with circuit MD optimization could theoretically yield better solutions, existing research has typically addressed automatic bootstrapping under the assumption of a predefined MD upper bound, effectively decoupling the two optimization problems due to their complexity [29,32]. Similarly, this work focuses exclusively on enhancing FHE circuit design, with bootstrapping operations falling beyond our scope.

2.3.4. *Significance of MC-Aware MD Miminization*

Achieving a lower MD at the expense of increased MC leads to a greater number of homomorphic multiplications, thereby affecting the overall computation time. It is therefore essential that MD reduction is performed with explicit consideration of MC to achieve effective FHE circuit optimization. This necessity highlights two key challenges: (1) the development of tailored circuit synthesis and optimization techniques capable of jointly optimizing both metrics, and (2) the formulation of a plausible cost metric — referred to as the *FHE cost* — that meaningfully captures the characteristics of the most efficient circuit implementation and can guide the synthesis process. This work is primarily dedicated to addressing the first challenge, namely, the design of an optimization framework that simultaneously considers both MD and MC, rather than treating them independently as in most existing algorithms.

The second challenge, the determination of the optimal form of the FHE cost metric, remains an open problem in the literature. While it is clear that the cost should primarily reflect the computational overhead of homomorphic multiplications, the precise quantitative impact of MD on overall performance has yet to be conclusively established. Empirical studies suggest that the best cost model may vary with the scheme, parameter set, and implementation [15]. For example, in [15], a power-law regression model was fitted to relate the multiplicative depth d of a circuit to the corresponding ciphertext size l (in kilobytes) in the BFV scheme, yielding $l = 1.2215 \cdot d^{2.0179}$. The asymptotic runtime complexity of homomorphic multiplication is also recognized as being comparable to that of arbitrary-precision integer multiplication, with bit complexity $O(n \cdot \log n \cdot \log\log n)$ [1].

Given these observations and for the sake of practical evaluation, we adopt $\mathrm{MC} \times \mathrm{MD}^2$ as the working form of the FHE cost in this paper. It is important to note, however, that our optimization framework is not tied to any fixed form of the FHE cost metric. Rather, it is designed with a flexible interface that allows users to specify an arbitrary cost function $\kappa(MC(G), MD(G))$ for a Boolean FHE circuit G (represented as an XOR-AND Graph (XAG)). This flexibility enables practitioners to tailor the optimization process to their preferred or empirically validated cost models, and supports future research aimed at identifying the most effective FHE cost formulations for different schemes, parameter regimes, or application contexts. Thus, our contribution provides not only an effective optimization method for jointly reducing MC and MD, but also a platform for further exploration of cost modeling in FHE circuit synthesis.

3. FHE-Cost-Optimum Synthesis for Boolean Functions

In this section, we focus on addressing the following problem: "Given a Boolean function with a limited number of inputs and a specified FHE cost metric that depends on both MC and MD, how can we exactly synthesize its FHE-cost-optimal circuit implementation?" This requires an exact synthesis formulation that supports specifying the target MC and MD of the XAG to be synthesized. Our approach leverages the concepts of the *AND fence* and the *abstract XAG*.

3.1. *Overview of the Methodology*

When synthesizing XAGs, our formulation is built upon the concepts of the AND fence and abstract XAG, associating each formulation with a specific use of AND gates in the target logic network. This subsection introduces how these concepts are adapted to enable our FHE-cost-optimal exact synthesis formulation.

3.1.1. *AND Fence*

An XAG s AND fence refers to the extracted information describing the use of AND gates in the logic network [42]. Based on our interest in the MC and MD of each XAG, we define the AND fence, denoted as $\mathcal{F}$, as a set representing the number of AND gates at each level of MD, symbolically:

$$\mathcal{F} = (c_1, c_2, \cdots, c_d),$$

where d is the MD of the XAG and each c_i within $\mathcal{F}$ is an integer, denoting the number of ANDs within the XAG whose MD is i. For example, the AND fence of the XAG in Fig. 1 is (1, 1). Based on an XAG's AND fence, its MC c can be calculated as:

$$c = \sum_{i=1}^{d} c_i. \tag{2}$$

For clarity, we attribute the MC and MD to an AND fence $\mathcal{F}$, defining them as the MC and MD of the XAG associated with $\mathcal{F}$. In symbolic terms, this can be expressed as *MD(*$\mathcal{F}$*)=*d and *MC(* $\mathcal{F}$*)=*c. As illustrated, the information provided by an XAG's AND fence is adequate for deducing both its MC and MD, and therefore, its FHE cost, without requiring additional information, such as node connections.

3.1.2. *Abstract XAG*

Abstract XAG is a general representation of XAG, in the sense that it removes information regarding the connections among XOR gates in an XAG. This is realized by using *XOR cloud*, an XOR gate whose fan-in size is flexible, allowing for any non-zero number of inputs [34], instead of being constrained to 2-input, as a network component. An XOR gate with a single input functions as a buffer.

In the original definition, an abstract XAG features that:

(a) Each fan-in of an AND gate is an XOR cloud;
(b) Each fan-in of an XOR cloud is either a PI or an AND gate in a lower *logic* level;
(c) Each PO is an XOR cloud.

In an abstract XAG, each step consists of a 2-input AND gate and its two fan-in XOR clouds. Thus, the number of steps in an abstract XAG numerically equals the number of AND gates, which is the MC c of the network. Symbolically, in an abstract XAG that implements an n-variable Boolean function f, each step x_i can be represented as

$$x_i = (\bigoplus_{x_j \in L_{i,1}} x_j) \wedge (\bigoplus_{x_j \in L_{i,2}} x_j),\ n < i \leq n + c,\ 1 \leq j < i, \tag{3}$$

where lists $L_{i,1}$ and $L_{i,2}$ indicate respectively the inputs of the two fan-in XOR clouds of the AND gate in the i-th step. The PO XOR cloud realizes the function f as a linear

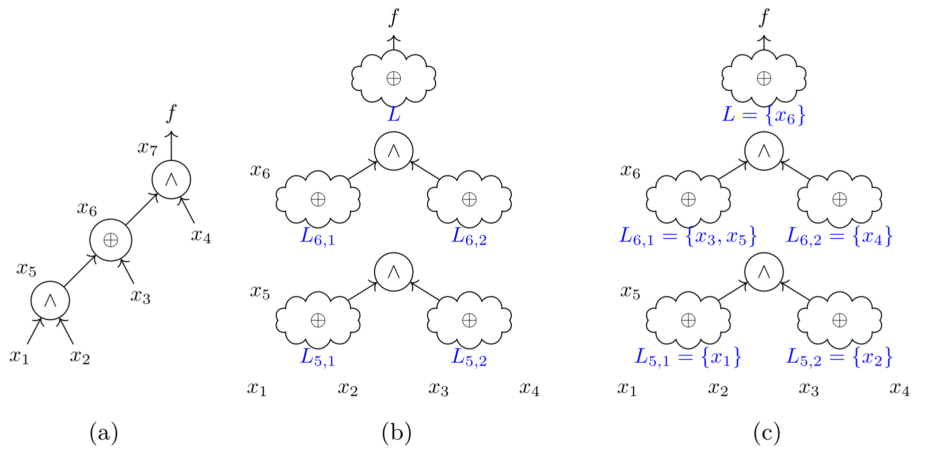

Fig. 2. Deterministically deriving the abstract XAG counterpart for an XAG in two steps: (a) Reproduction of Fig. 1; (b) First step: Determining the topology of the abstract XAG based on the XAG's AND fence; (c) Second step: Configuring the fan-ins of each XOR clouds within the abstract XAG according to the node connection in the XAG.

function over a set of PIs and steps: denoting L as the inputs of the PO XOR cloud,

$$f = \bigoplus_{x_i \in L} x_i, \ 1 \leq i \leq n + c. \tag{4}$$

Building upon the original definition, which is inherently aware of the MC of the circuit, we enhance the awareness of MD in each step by introducing additional constraints on the feasible fan-ins of each XOR cloud within a step x_i. This is achieved through overwriting the aforementioned feature (b) with the following two feature descriptions:

i Any fan-in of an XOR cloud is either a PI or a step at a *level of MD* lower than the current step, i.e., $\forall x_j \in L_{i,1} \cup L_{i,2} : \sigma_j < \sigma_i$;
ii Among the fan-ins of the two XOR clouds in the current step, at least one should be a step in the preceding level of MD, i.e., $\exists x_j \in L_{i,1} \cup L_{i,2} : \sigma_j = \sigma_i - 1$.

Through the redefinition, a direct mapping is established between an AND fence $\mathcal{F}$ and the *topology* of an abstract XAG, thereby extending the definition of AND fence from XAGs to abstract XAGs. Given an XAG, whose AND fence is $\mathcal{F}$, its functionally equivalent abstract XAG, whose AND fence is also $\mathcal{F}$, can be identified by configuring the fan-ins of the XOR clouds within the topology, as illustrated in the following example.

Example 1. Fig. 2 provides an example illustrating the process of deriving the abstract XAG counterpart for an XAG, with the fan-ins of each XOR cloud explicitly specified beneath it in blue. Given the AND fence of the XAG in Fig. 2a, denoted as $\mathcal{F} = (1, 1)$, the MD of steps x_5 and x_6 are determined to be $\sigma_5 = 1$ and $\sigma_6 = 2$, respectively. This indicates that the topology of the abstract XAG resembles the depiction in Fig. 2b. By configuring the fan-ins of each XOR cloud correspondingly, an abstract XAG functionally equivalent to the XAG is derived in Fig. 2c.

While the conversion of an XAG to an abstract XAG, as depicted in Fig. 2, is deterministic, the reverse process is not. A straightforward approach to converting an abstract XAG into an XAG involves decomposing XOR clouds into 2-input XOR gates, though this may result in using more XOR gates than the minimum required in the resulting XAG. However, since our FHE cost metric is determined by the MC and MD of an XAG — both of which are preserved during decomposition — if an abstract XAG implementation is FHE-cost-optimum, the XAG obtained through XOR cloud decomposition will also be FHE-cost-optimum. Therefore, the FHE-cost-optimum XAG implementation of a Boolean function can be generated by first finding its FHE-cost-optimum abstract XAG implementation, which can be efficiently solved by formulating it as a *Boolean satisfiability* (SAT) problem.

It is important to note that, given an identical AND fence, the abstract XAG counterpart of an XAG is not necessarily unique. In other words, for a given XAG, there may exist multiple configurations of fan-ins within the XOR clouds in the same abstract XAG topology that allow the resulting abstract XAG to implement the same Boolean function as the XAG. These various abstract XAGs will have the same FHE cost, as they share an identical AND fence. This observation highlights the equivalence between the following two tasks: Given a Boolean function f and an AND fence $\mathcal{F}$, (1) determining whether there exists an XAG associated with $\mathcal{F}$ that implements f, and (2) ascertaining whether there exists a configuration of fan-ins of the XOR clouds within an abstract XAG topology adhering to $\mathcal{F}$, such that the resulting abstract XAG implements f. A detailed introduction to our SAT formulation for the latter task will be provided later.

3.1.3. *Sketch of Methodology*

Building upon our redefined concepts of AND fence and abstract XAG, we derive the following insights:

(1) An XAG's FHE cost can be derived from its AND fence.
(2) Leveraging SAT solving, we can efficiently determine whether there exists an abstract XAG implementation for a target function while satisfying a specified AND fence.
(3) An abstract XAG can always be converted into an XAG by decomposing XOR clouds, and this process preserves FHE cost.

These observations collectively shape our methodology for synthesizing the FHE-cost-optimum XAG implementation for Boolean functions. By enumerating AND fences in an FHE-cost-ascending manner (leveraging (1)), with each enumeration treated as a SAT problem (drawing on (2)), the FHE-cost-optimum abstract XAG implementation can be identified, which, in turn, translates to the FHE-cost-optimum XAG implementation, indicated by (3).

3.2. *SAT Encoding*

We introduce the requisite variables and clauses, and subsequently encode the previously posited decision problem into a SAT problem.

3.2.1. *Variables*

Our encoding is based on two types of binary variables: (1) *selection variables*, which encode the fan-in configurations of XOR clouds within an abstract XAG, and (2) *function variables*, which encode the function computed at each step.

Depending on whether an XOR cloud is a fan-in in a step or is the PO, selection variables can be further classified, with each denoted as:

i) s_{ijk}, where $n < i \leq n + c$, $1 \leq j < i$, and $k \in \{1, 2\}$, whose being *true* indicates that "x_j (a PI when $j \leq n$, otherwise a step) connects to the k-th fan-in XOR cloud within step x_i," i.e., $x_j \in L_{i,k}$.
Ii) s_j, for $1 \leq j \leq n+c$, whose being *true* signifies that "x_j connects to the PO XOR cloud," i.e., $x_j \in L$.

We denote each function variable as f_{jl}, for $1 \leq j \leq n + c$ and $0 \leq l \leq 2^n$. With $(b_n, \cdots, b_1)_2$ being the binary representation of l, variable f_{jl}'s evaluating to *true* indicates that "PI x_j is assigned to b_j" for $j \leq n$, or "step x_j produces *true* given the PI assignment: $x_1 = b_1$,$\cdots$,$x_n = b_n$" for $j > n$.

3.2.2. *Clauses*

Our encoding includes five types of clauses.

For each step x_i, i.e., $n < i \leq n + c$, *Clause 1* ensures the correctness of the Boolean operations involved in this step:

$$f_{il} \leftrightarrow \bigwedge_{k\in\{1,2\}} \bigoplus_{j=1}^{i-1} (s_{ijk} \wedge f_{jl}).$$

For each PI x_i, where $1 \leq i \leq n$, $f_{il} = b_i$, *Clause 2* ascertains that the resulting network implements the target function f:

$$f(b_1, \cdots, b_n) = \bigoplus_{j=1}^{n+c} (s_j \wedge f_{jl}).$$

In Clause 2, the two sides are connected by '$=$' instead of '$\leftrightarrow$'. This choice is deliberate, as the left-hand side signifies the fixed output of function f under the input assignment: $x_1 = b_1$,$\cdots$,$x_n = b_n$, which is a constant rather than a variable.

Clause 3 guarantees that the fan-in size of each XOR cloud, whether within a step or serving as the PO, must be at least one:

$$\bigvee_j s_{ijk},\ n < i \leq n + c,\ 1 \leq j < i,\ k \in \{1, 2\}$$

and

$$\bigvee_j s_j,\ 1 \leq j \leq n + c.$$

Clause 4 implements feature (b)-i, introduced in Section 3.1.2, as:

$$\bigwedge_{j,k} \overline{s_{ijk}},\ n < i \le n + c,\ \min\{j' \mid \sigma_{j'} = \sigma_i\} \le j < i,\ k \in \{1, 2\}.$$

Clause 5 incorporates feature (b)-ii as:

$$\bigvee_{j,k} s_{ijk},\ n < i \le n + c,\ \min\{j' \mid \sigma_{j'} = (\sigma_i - 1)\} \le j \\ \le \max\{j' \mid \sigma_{j'} = (\sigma_i - 1)\},\ k \in \{1, 2\}.$$

Clauses 4 and 5 ensure the abstract XAG topology aligns with AND fence $\mathcal{F}$.

The clauses, if not already in *conjunctive normal form* (CNF), undergo additional conversion into CNF formulas using the *Tseitin encoding* [38] before being resorted to SAT solving. Using a SAT solver, we can determine whether there exists an abstract XAG with AND fence $\mathcal{F}$ that implements Boolean function f by solving the encoded SAT instance.

3.3. *Identification of AND Fence Candidates*

In this sub-section, we detail our strategy for identifying which AND fence candidates shall be investigated when synthesizing the FHE-cost-optimum circuit implementation for a given Boolean function. This pursuit aligns with our goal sketched in Section 3.1.3: To ensure the optimality of the synthesized circuit, all potentially viable AND fences must be considered as candidates. We ensure that all such AND fence candidates are included by determining the ranges within which the MC and MD of the FHE-cost-optimum circuit may fall.

The goal of this section is to synthesize the optimum circuits for all up to n-variable Boolean functions. However, the number of functions grows double-exponentially as the number of variables increases. For instance, with $n = 5$, there are 2^{32} distinct functions, making it impractical to target each one individually. To overcome this challenge To address this, we apply *affine function classification* [19], a *Boolean classification* technique, to effectively reduce the problem s complexity. A detailed introduction to affine function classification is given in Appendix A.2. Given the invariance of affine-equivalent operations concerning MC, the MC-minimum XAG implementation of a Boolean function f can be derived from the MC-minimum XAG that represents its affine equivalence class [37]. All 5-variable Boolean functions can be categorized into 48 affine equivalence classes [36], with the MC-minimum XAG implementation for each representative function already identified in prior research [34]. We further note that the MD of an XAG obtained in this way also remains consistent with its representative XAG, formalized as

Theorem 1. *Affine-equivalent operations are MD-preserving if the input variables share an identical MD.*

Proof. The proof of this theorem is provided in Appendix A.3. □

Algorithm 1: Synthesizing the FHE-cost-optimum XAG implementations for Boolean functions

```
Input: Boolean function $f$; MC $c_r$ and MD $d_r$ of the MC-minimum XAG for $f$.
Output: FHE-cost-minimum XAG implementation for $f$, $N$.
$d \leftarrow d_r$
known-minimum FHE cost $\kappa \leftarrow c_r \cdot d_r^2$
while $d > 1$ do
    $d \leftarrow d - 1$
    $c_{max} \leftarrow \left\lfloor \frac{\kappa}{d^2} \right\rfloor$
    $keep\_searching \leftarrow false$
    for $c \leftarrow c_r$ to $c_{max}$ do
        foreach $\mathcal{F} \in \mathcal{F}(c, d)$ do
            SAT instance $\zeta \leftarrow SAT\_encoding(f, \mathcal{F})$
            abstract XAG $N' \leftarrow SAT\_solving(\zeta)$
            if $N' \neq NULL$ then
                $\kappa \leftarrow c \cdot d^2$
                $keep\_searching \leftarrow true$
                break
        if $keep\_searching$ then break
    if not $keep\_searching$ then
        XAG $N \leftarrow decompose\_XOR\_clouds(N')$
        return $N$
```

Therefore, for any 5-variable Boolean function f, there always exists an MC-minimum implementation, the MC and MD of which is known, denoted as c_r and d_r. Thus, the FHE-cost-optimum implementation of f — unless identical to the MC-minimum one — has MC and MD values c and d that satisfy the conditions

$$c \geq c_r, d < d_r, \text{and } c \cdot d^2 < c_r \cdot d_r^2.$$

Given target MC c and MD d, we denote the complete set of AND fence candidates that satisfy Eq. 2 as $\mathcal{F}(c, d)$. Finding all AND fences $\mathcal{F} \in \mathcal{F}(c, d)$ is essentially a *positive integer partition* problem, a well-studied topic in number theory and combinatorics with detailed solutions in the literature [27]. By considering all AND fences meeting the specified conditions in the synthesis procedure, we ensure the minimum FHE costs in the synthesized implementations.

3.4. *Exact Synthesis Paradigm for Boolean Functions*

Alg. 1 details our paradigm for synthesizing the FHE-cost-optimum XAG implementations for Boolean functions. Commencing with the known MC-minimum implementation of the target Boolean function, we systematically investigate the existence of an abstract XAG whose MD d is slightly smaller than the known solution, i.e., $d = d_r - 1$. AND fence candidates with the same MD d are ordered in an MC-ascending manner, so that lower-FHE-cost candidates are considered first. If an abstract XAG is successfully synthesized using an AND fence with the currently targeted MD, the exploration extends to AND fences with a rather smaller MD. Termination of the exploration occurs if none of the AND fence candidates with the MD of d leads to a valid implementation. In such cases, it indicates either (a) If $d < d_r - 1$, the previously synthesized abstract XAG, with an MD of $d + 1$, is deemed FHE-cost-optimum; or (b) The known MC-minimum implementation stands as FHE-cost-optimum.

Intuitively, the FHE cost should be the determinant guiding the investigation of AND fences, rather than MD, to prioritize the exploration of lower-FHE-cost candidates, as sketched in Section 3.1.3. However, Algo. 1 is structured to conduct the exploration based on MD. This strategy is devised based on the observation that aggressively initiating the exploration from the FHE-cost-minimum AND fence candidate tends to introduce a considerable number of unsatisfiable SAT instances into the paradigm, leading to an inefficient synthesis procedure.

The following observation provides evidence that the strategy does not compromise the optimality of the synthesized XAG.

Theorem 2. *If no abstract XAG exists for a Boolean function f with any fence $\mathcal{F}_1$ of MD d, then no abstract XAG exists for f with any fence $\mathcal{F}_2$ of MD $d - 1$.*

Proof. The proof of this theorem is provided in Appendix B. □

4. Exact Synthesis for Sub-circuits

While the exact synthesis approach proposed in Section 3 ensures the optimality of the circuits synthesized for Boolean functions, scalability concerns limit its application to small-scale functions with a restricted number of variables. To deliver high-quality solutions for practical functions, we shift our focus from finding the optimum circuit for a *function* to determining the optimum implementation for a *cut*, i.e., sub-circuit. Leveraging exact synthesis to replace each sub-circuit within a baseline circuit with its optimum implementation, the quality of the entire circuit is improved ultimately.

4.1. *Impacts of Non-zero Input MD*

To ultimately minimize the FHE cost of the entire circuit, our strategy is to optimize the implementation of each cut rooted on the critical path. Specifically, for each such cut, we regard the circuit that can maximally minimize the MD of its root node as the optimal implementation for this cut. This is because, with a local view, it is difficult to estimate the contribution of a cut's MC to the entire circuit's MC, because of potential logic sharing. However, this is not against our proposal that FHE circuit optimization shall be MC-aware. To that end, implementations that require a significantly increased MC are excluded from our exploration by considering only the AND fence candidates collected in Section 3.3.

However, when considering the local function of a cut — defined as the Boolean function realized by the root node with respect to the leaf nodes — using the exact synthesis paradigm proposed in Section 3.4 to generate the circuit that optimally implements this local function does not guarantee a locally optimal implementation for the cut, in terms of minimizing the MD of the root node. This is because the leaves of the cut likely have

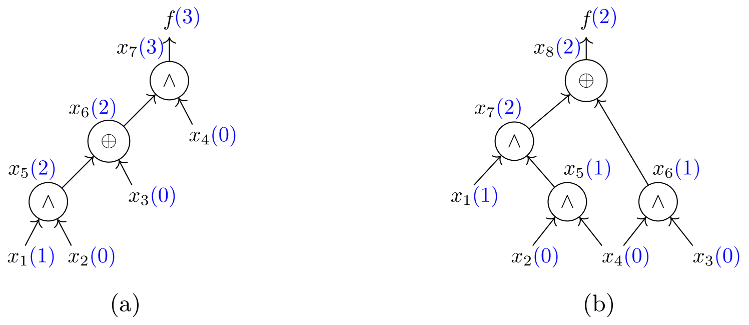

Fig. 3. Two XAG implementations for function #7800, under input MD $\mathcal{L} = (1, 0, 0, 0)$, with each node's MD remarked in blue: (a) XAG N_1; (b) XAG N_2.

non-zero MD. When synthesizing the optimal implementation for a cut, the leaves of the cut are likely logic gates (i.e., internal nodes) of a logic network that implements the entire computation, rather than PIs. Therefore, the problem addressed in this section, "How to synthesize the optimal[3] implementation for a cut," is a generalization of the problem tackled in Section 3, "How to exactly synthesize the optimum implementation for a Boolean function." The problem in Section 3 can be regarded as a special case of the current one, with the constraint that the MD of the leaves is always zero.

Regarding an n-cut-highlighted sub-circuit, we refer to the MD of the leaves as the *input MD* of the sub-circuit, denoted as an n-element set $\mathcal{L}$, whose i-th element represents the MD of the i-th leaf of the cut. $\mathcal{L}$ is defined as *balanced* if all its elements are identical; Otherwise, it is *imbalanced*.

The optimal implementation of a cut depends on its input MD. Even when targeting the same local function, the optimal circuit implementation of a cut varies based on the input MD. We illustrate this observation in the following example.

Example 2. The two XAGs depicted in Fig. 3, denoted as N_1 and N_2, both realize a 4-cut $\mathcal{C}$, characterized by the local function f represented in truth table #7800. The AND fences associated with N_1 and N_2 are designated as $\mathcal{F}_1 = (1, 1)$ and $\mathcal{F}_2 = (2, 1)$, respectively. Notably, it is established that N_1 exhibits an MC of 2 and an MD of 2, whereas N_2 is characterized by an MC of 3 and an MD of 2. N_1 stands as the more FHE-cost-efficient implementation of *Boolean function* f, given its lower MC, therefore, lower FHE cost. However, a nuanced consideration arises when examining the input MD of this cut, denoted as $\mathcal{L} = (1, 0, 0, 0)$. From Fig. 3, it is evident that the MD of the roots (x_7 in Fig. 3a and x_8 in Fig. 3b) are 3 and 2, respectively. This observation points out that, in the context of synthesizing sub-circuits, N_2 has the potential to serve as a better implementation than N_1 for *cut* $\mathcal{C}$.

[3]We refer to the implementation synthesized for each cut as optimal rather than optimum because our approach excludes choices that could reduce the MD of the root node by allowing a significant local increase in MC. However, these excluded choices might occasionally result in lower-FHE-cost circuit designs, as a locally increased MC does not necessarily correspond to a globally increased MC due to potential logic sharing among different cuts.

This observation necessitates incorporating the input MD of cuts into the SAT encoding to achieve the optimal implementation of each cut. Our investigation begins by establishing a symbolic analysis to examine the impact of a cut's input MD on the divergence between "the optimum implementation of a Boolean function f" and "an optimal implementation of a cut $\mathcal{C}$, whose local function is f."

To provide a local perspective, we define the *local MD* of a node within an XAG that implements cut $\mathcal{C}$ as the XAG's MD d minus the maximum number of AND nodes on any path from this node to the root node, denoted as δ. The local MD of the root node of cut $\mathcal{C}$, denoted as δ_{root}, numerically equals d, the MD of the XAG. To further enhance the intuitiveness of our analysis, we leverage abstract XAG as the logic representation. Since the conversion from an XAG N to its abstract counterpart N' is deterministic, as depicted in Fig. 2, generality is ensured in our analysis.

Let cut $\mathcal{C}$ has n leaves, x_1 to x_n, and an input MD of $\mathcal{L} = (\sigma_1, \cdots, \sigma_n)$. An c-step abstract XAG, whose topology is characterized by an AND fence $\mathcal{F} = (c_1, \cdots, c_d)$, implements cut $\mathcal{C}$. The relationship between $\mathcal{F}$ and c adheres to Eq. 2.

Denoting the difference between the i-th leaf node x_i's local MD δ_i and the root node's local MD δ_{root} as Δ_i, we have:

$$\sigma_{root} = \max\{\sigma_i + \Delta_i \mid 1 \leq i \leq n\}. \tag{5}$$

According to our definition of local MD, Δ_i represents the maximum number of AND nodes on any path from leaf node x_i to the root node, which corresponds to the maximum number of steps on any path from leaf node x_i to the root node when considering the abstract XAG as the network representation. The value of Δ_i must satisfy $0 \leq \Delta_i \leq d$. The lower bound is reached when a leaf x_i exclusively serves as a fan-in of the PO XOR cloud, not contributing to any other XOR clouds within the abstract XAG N'; The upper bound is achieved if a leaf x_i contributes to a step whose local MD is one. While the definition of abstract XAGs guarantees the upper bound, the lower bound is not always achievable. Consider Eq. 5, when the input MD $\mathcal{L}$ is balanced, i.e., when $\sigma_1 = \cdots = \sigma_n = a$, there is a strong correlation between σ_{root} and δ_{root}, which can be symbolically summarized as below:

$$\begin{aligned} \sigma_{root} &= a + d = a + \delta_{root} \\ &= a + MD(N') = a + MD(\mathcal{F}). \end{aligned} \tag{6}$$

Examining Eq.6 reveals that when $\mathcal{L}$ is balanced, σ_{root} is collectively influenced by $\mathcal{L}$ and the AND fence $\mathcal{F}$, obviating the need to identify the critical path.

Conversely, in cases of imbalanced input MD, a meticulous examination of each path becomes essential to identify the critical path determining σ_{root}, according to Eq. 5. This necessitates a detailed SAT encoding, extending to the one proposed in Section 3.2.

4.2. *Integrating Scheduling into SAT Encoding*

To address the challenges posed by imbalanced input MD and ensure the synthesis of optimal implementations under these intricacies, we introduce an extended SAT encoding. In this extended encoding, each SAT instance corresponds not only to a specific

AND fence but also to a particular value assignment for Δ_i for $1 \leq i \leq n$, for the target n-cut. Following this setup, every SAT instance is linked to a distinct σ_{root}, allowing us to leverage the core concept of Algo. 1 — The iterative solution of a set of SAT instances ensures that the first satisfiable instance corresponds to the optimal implementation of the target cut.

Based on the definition of local MD, for a leaf x_i, the value assignment "$\Delta_i = a$, where $0 \leq a \leq d$, " is equivalent to constraining that "the lowest level of local MD at which x_i contributes to, is $d - a$," where d is the MD of the target AND fence. In simpler terms, determining the value of Δ_i essentially involves scheduling when leaf x_i becomes available to serve as a fan-in for a step at that particular level of local MD. Therefore, a specific value assignment to Δ_i for $1 \leq i \leq n$ can be viewed as a *scheduling*, denoted as

$$\mathcal{S} = (\Delta_1, \Delta_2, \cdots, \Delta_n).$$

A scheduling solution can be incorporated into the SAT encoding as an additional clause type concerning the selection variables. By definition, a leaf x_j cannot serve as a fan-in for any step with a local MD lower than $d - \Delta_j$. Therefore, let $x_{i'}$ represent the last step with a local MD lower than $d - \Delta_i$, i.e., $\delta_{i'} = (d - \Delta_i - 1)$ and $\delta_{i'+1} = (d - \Delta_i)$. Then, the scheduling on leaf x_j is ensured by the following constraint:

$$\bigwedge_{i,k} \overline{s_{ijk}}, \ 1 \leq i \leq i', \ k \in \{1, 2\}.$$

4.3. *Strategic Selection of Scheduling Solutions*

With the proposed encoding accommodating a scheduling solution atop a given AND fence, the question arises: "Which scheduling solutions warrant investigation?"

The naïve approach to finding the σ_{root}-minimal implementation for an n-cut $\mathcal{C}$ under a given AND fence $\mathcal{F}$ involves enumerating all $(d+1)^n$ scheduling solutions. However, this exhaustive exploration is inefficient. To address the inefficiency, we propose a strategic selection of promising scheduling solutions, consisting of two key components: (1) Identifying the initial scheduling solution that represents the theoretical minimum σ_{root} under the AND fence $\mathcal{F}$. (2) Identifying the subsequent scheduling solution for exploration, when no feasible implementation exists under the currently investigated scheduling solution.

Without loss of generality, throughout this section, we assume the input MD $\mathcal{L}$ of cut $\mathcal{C}$ satisfies $\sigma_1 \leq \cdots \leq \sigma_n$ for clarity.

4.3.1. *Identification of the Initial Scheduling Solution*

Intuitively, the initial scheduling solution instructs each leaf node to contribute to a step as late as possible, to minimize the MD of the root node. However, we observe that this decision-making process must consider the following rule: for each level of local MD, there should be a sufficient number of fan-in candidates — either leaves or steps from lower levels of local MD — for the steps at the current level to select from.

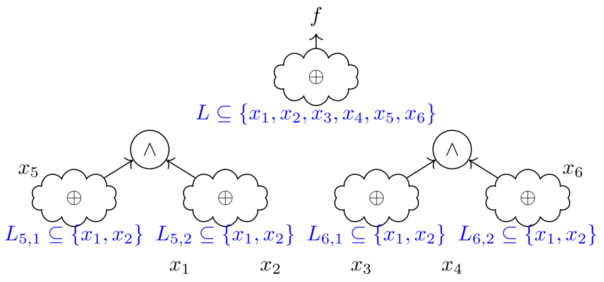
f
$\oplus$
$L \subseteq \{x_1, x_2, x_3, x_4, x_5, x_6\}$
$\wedge$
$\wedge$
x_5
x_6
$\oplus$
$\oplus$
$\oplus$
$\oplus$
$L_{5,1} \subseteq \{x_1, x_2\}$
$L_{5,2} \subseteq \{x_1, x_2\}$
$L_{6,1} \subseteq \{x_1, x_2\}$
$L_{6,2} \subseteq \{x_1, x_2\}$
x_1
x_2
x_3
x_4

Fig. 4. A visualization of the SAT instance corresponding to the synthesis problem described in Example 3.

Example 3. Consider a scenario where we explore the optimal implementation of a 4-cut $\mathcal{C}$, with local function f and input level $\mathcal{L} = (0, 0, 0, 1)$, using an AND fence $\mathcal{F}_1 = (2)$. The topology described by $\mathcal{F}_1$ outlines the abstract XAG to be synthesized, featuring two steps, x_5 and x_6, both with local MD $\sigma_5 = \sigma_6 = 1$. Assume the initial scheduling solution is set as $\mathcal{S}_1 = (1, 1, 0, 0)$ — meaning steps x_5 and x_6 are restricted to selecting fan-ins only from leaves x_1 and x_2. A visualization of the corresponding SAT instance is given in Fig. 4. This leads to two possible consequences: (1) Both steps x_5 and x_6 implement the same function $x_1 \wedge x_2$. (2) One of the two steps trivially implements a linear function over $\{x_1, x_2\}$. In other words, the satisfiability of the current SAT instance is equivalent to one with the same setup but with an AND fence $\mathcal{F}_2 = \{1\}$. Since $\mathcal{F}_2$ exhibits a lower MC compared to $\mathcal{F}_1$, the exploration of which must has already been addressed. Hence, the current SAT instance is determined to be unsatisfiable without the need for SAT solving. Judiciously selecting the initial scheduling solution as $\mathcal{S}_2 = (1, 1, 1, 0)$ can bypass this trivial case.

We identify a reasonable initial scheduling solution by addressing the following question: "Given a certain number of fan-in candidates to select from, how many non-linear functions can a step within an abstract XAG implement?" Recall the general representation of the function implemented by an arbitrary step x_i given by Eq. 3. Our calculation is based on considering all potential configurations of $L_{i,1}$ and $L_{i,2}$. Among them, cases where $L_{i,1} = L_{i,2}$ are first excluded, as they lead to linear functions, whose realization does not require any AND nodes. We then exclude two cases that result in repetitive functions: (1) Due to the commutativity of the AND operation, functions realized by exchanging $L_{i,1}$ and $L_{i,2}$ are identical; (2) For any function realized with $L_{i,1}$ and $L_{i,2}$, where either $L_{i,1} \subseteq L_{i,2}$ or $L_{i,2} \subseteq L_{i,1}$, there always exists an implementation where $L_{i,1} \nsubseteq L_{i,2}$ or $L_{i,2} \nsubseteq L_{i,1}$ [16]. Applying the rules outlined above, we calculate the number of non-linear Boolean functions a step can realize. While only cases with up to 5 fan-in candidates are summarized in Table 1, the proposed calculations are applicable beyond this limit.

After scheduling the availability of several leaf nodes to ensure that each step has sufficient fan-in candidates, the remaining unscheduled leaf nodes are then scheduled to be available at the lowest possible level of local MD, following the so-called *as soon as possible* (ASAP) scheduling, provided they do not form a unique critical path. Assuming leaves x_1 to $x_{j'}$ are already scheduled, the scheduling for an unscheduled leaf x_i, where

Table 1. Number of non-linear functions a step within an Abstract XAG can implement.

#fan-in candidates	2	3	4	5
#non-linear functions	1	9	55	285

Algorithm 2: Selecting the initial scheduling solution for an AND fence

```
Input: AND fence $\mathcal{F} = (c_1, \cdots, c_d)$; Input MD $\mathcal{L} = (\sigma_1, \cdots, \sigma_n)$.
Output: Scheduling solution $\mathcal{S} = (\Delta_1, \cdots, \Delta_n)$; MD of the root node $\sigma_{root}$.
1  set of indices of unscheduled leaf nodes $s \leftarrow \{1, \cdots, n\}$
2  number of available fan-in candidates $\#cand \leftarrow 0$
3  for $i \leftarrow 1$ to $d$ do
4      number of leaf nodes to schedule $\#leaves \leftarrow look\_up(\sum_{i'=1}^{i} c_{i'}) - \#cand$
5      if $\#leaves > 0$ then
6          set of indices of leaf nodes to schedule $s' \leftarrow$ the first $\#leaves$ elements in $s$
7          foreach $j \in s'$ do
8              $\Delta_j \leftarrow i$
9              $\#cand \leftarrow \#cand + 1$
10         $\#cand \leftarrow \#cand + c_i$
11 $j' \leftarrow n - |s|$
12 $\sigma_{root} \leftarrow \max\{\sigma_j + \Delta_j \mid 1 \leq j \leq j'\}$
13 foreach $i \in s$ do
14     $\Delta_i \leftarrow \sigma_{root} - \sigma_i$
15 return $\{\mathcal{S}, \sigma_{root}\}$
```

$j' < i \leq n$, should adhere to the following inequality:

$$\sigma_i + \Delta_i \leq \max\{\sigma_j + \Delta_j \mid 1 \leq j \leq j'\}.$$

Determining Δ_i in this manner is preferable because the resulting SAT instance covers the scenarios where a leaf node x_i is scheduled to higher levels of local MD, thereby avoiding repetitive SAT solving.

The two steps for our initial scheduling solution selection are outlined in Algo. 2. The function *look_up* in line 4 determines, based on the number of steps, the required number of fan-in candidates obtained from Table 1. These candidates include both leaves scheduled to be available at the current level of local MD (line 8) and steps belonging to lower levels of local MD (line 10).

4.3.2. *Identification of the Subsequent Scheduling Solution*

When a SAT instance turns out to be unsatisfiable, it implies there does not exist an abstract XAG, whose topology adheres to the currently investigated AND fence $\mathcal{F}$, that implements the target cut $\mathcal{C}$, under the current scheduling solution $\mathcal{S}_1$. Therefore, to find such a $\mathcal{F}$-based implementation, if it does exist, we have to relax the expectation on the resulting MD of the root of $\mathcal{C}$, by adjusting $\mathcal{S}_1$ to make the leaves accessible in earlier levels of local MD.

```
Algorithm 3: Synthesizing FHE-cost-optimal XAG implementations for cuts
Input: Cut $\mathcal{C}$; Current XAG implementation of cut $\mathcal{C}$, $N_{old}$;
        Library of AND fence candidates $lib$.
Output: Optimal XAG implementation for $\mathcal{C}$, $N$.
1  $N \leftarrow exact\_synthsis\_for\_function(\mathcal{C}.f)$
2  If $is\_balanced(\mathcal{C}.\mathcal{L})$ then return $N$
3  $\sigma_{root} \leftarrow calculate\_root\_MD(N,\mathcal{C}.\mathcal{L})$
4  foreach $\mathcal{F} \in lib$ do
5      $\{\mathcal{F}.\mathcal{S}, \mathcal{F}.\sigma_{root}\} \leftarrow select\_initial\_scheduling(\mathcal{F}, \mathcal{C}.\mathcal{L})$
6  target MD of root node $\sigma_{target} \leftarrow \min\{\mathcal{F}.\sigma_{root} \mid \mathcal{F} \in lib\}$
7  while $\sigma_{target} < \sigma_{root}$ do
8      foreach $\mathcal{F} \in lib$ do
9          if $\mathcal{F}.\sigma_{root} = \sigma_{target}$ then
10             SAT instance $\zeta \leftarrow extended\_SAT\_encoding(\mathcal{C}.f,\mathcal{F})$
11             abstract XAG $N' \leftarrow SAT\_solving(\zeta)$
12             if $N' \neq NULL$ then
13                 XAG $N \leftarrow decompose\_XOR\_clouds(N')$
14                 return $N$
15             else
16                 $\{\mathcal{F}.\mathcal{S}, \mathcal{F}.\sigma_{root}\} \leftarrow update\_scheduling(\mathcal{F})$
17     $\sigma_{target} \leftarrow \sigma_{target} + 1$
18 return N
```

If all elements in $\mathcal{S}_1$ are d, indicating all leaves are already scheduled to the lowest level, it means there is no $\mathcal{F}$-based circuit implementation for cut $\mathcal{C}$. Otherwise, each Δ_i in the subsequent scheduling solution $\mathcal{S}_2 = (\Delta_1, \cdots, \Delta_n)$ is determined to be $\min\{d, \sigma_{root} + 1 - \sigma_i\}$, where σ_{root} is the previously expected MD of the root of cut $\mathcal{C}$, calculated with $\mathcal{S}_1$ as the scheduling solution. Given the algorithmic similarity between determining the subsequent scheduling solution and the second step of identifying the initial scheduling solution (lines 12-14 in Algo. 2), the corresponding pseudocode is omitted.

4.4. *Exact Synthesis Paradigm for Sub-circuits*

Algo. 3 outlines our exact synthesis paradigm for synthesizing optimal implementations for sub-circuits.

The exact synthesis paradigm for function (Algorithm 1) is initially invoked to generate a baseline implementation for the target cut. This implementation is guaranteed to be optimal under the condition of a balanced input MD (line 2). The baseline implementation establishes a known lowest MD for the root of the cut, which serves as an upper bound for the target MD in the paradigm (line 7). For each AND fence candidate, its initial scheduling solution, along with the corresponding lowest expected MD of the root, is obtained by applying Algorithm 2. The optimality of the solution is ensured by prioritizing the investigation of AND fence candidate with the potential to achieve the lowest expected MD of the cut's root. If it turns out to be infeasible, its scheduling solution and its lowest expected MD of the root are updated (line 16).

4.5. *Classifying Exact Synthesis Queries*

Optimizing the FHE cost of a logic network consists of a series of queries of optimally synthesizing sub-circuits, each of which is handled by invoking the exact synthesis paradigm (Algo. 3). However, some queries may share the same optimal solutions. By identifying exact synthesis queries with identical optimal solutions, the synthesized solutions can be reused to avoid repetitive SAT solving, improving the efficiency of optimizing the circuit. To that end, we address the following question: "Under what conditions do two cuts share identical optimal implementations?"

Aligning with our strategy for optimally synthesizing circuits for functions as detailed in Section 3, the Boolean classification technique is utilized. However, recognizing that not all affine-equivalent operations preserve MD under an imbalanced input MD, we have opted for *NPN classification* [24], as NPN-equivalent operations reliably maintain MD (see Appendix A.1 for an in-depth introduction to NPN classification). Through NPN classification, two cuts share the same optimal implementation if: (1) their local functions belong to the same NPN equivalence class, and (2) their input MD aligns after reordering elements following the *NPN canonicalization* process.

Moreover, we found that the second requirement can be generalized by generating a *signature* for the input MD of a cut and basing the identification on signatures. Deriving a signature from the input level of a cut involves two distinct steps.

Lemma 1. (Alignment)
Let $\mathcal{C}_1, \mathcal{C}_2$ be n-cuts with the same local function and input MDs $\mathcal{L}_1 = (\sigma_1, \ldots, \sigma_n)$ and $\mathcal{L}_2 = (\sigma'_1, \ldots, \sigma'_n)$. If $\sigma_i - \sigma'_i$ is a constant a for all i, then the two cuts have identical optimal implementations.

Proof. The proof of this lemma is provided in Appendix C.1. □

Lemma 1 serves as the first step of the derivation of signatures, termed *alignment*. This step involves subtracting the minimum element σ_1 from each element in $\mathcal{L}$.

Lemma 2. (Dominance)
Let $\mathcal{C}_1$ be an n-cut with input MD $\mathcal{L}_1 = (\sigma_1, \ldots, \sigma_n)$ (sorted) and optimal XAG MD d. If there exists i with $\sigma_i - \sigma_{i-1} \geq d$, then the optimal implementation for $\mathcal{C}_1$ is also optimal for any n-cut $\mathcal{C}_2$ with the same local function and input MD $\mathcal{L}_2 = (0, \ldots, 0, \sigma_{i+1} - \sigma_i, \ldots, \sigma_n - \sigma_i)$.

Proof. The proof of this lemma is provided in Appendix C.2. □

Lemma 2 suggests that in the presence of a substantial *gap* among the MD of the target cut's leaves, attention should be directed only to those with the potential to form the critical path. The remaining leaves can be treated as if their MD is zero since they cannot be part of the critical path. This insight gives birth to the *dominance* step.

The two steps jointly facilitate deriving a signature of the input MD of a cut, as outlined in Algo. 4. The configuration of the parameter θ depends on n, the size of the target cut. For example, when targeting 5-cuts, θ is set to 3 because the MD of AND fence

Algorithm 4: Deriving the signature of an input MD

```
Input: Input MD of the target $n$-cut, $\mathcal{L} = (\sigma_1, \cdots, \sigma_n)$; Threshold $\theta$.
Output: Signature of $\mathcal{L}$, $sig$.
1 $sig \leftarrow \mathcal{L}$
2 foreach $\sigma \in sig$ do
3     $\sigma \leftarrow \sigma - \sigma_1$                          ▷ Alignment
4 for $i \leftarrow n$ to 2 do
5     if $\sigma_i - \sigma_{i-1} \geq \theta$ then
6         foreach $\sigma \in sig$ do
7             $\sigma \leftarrow \max\{0, \sigma - \sigma_i\}$       ▷ Dominance
8         break
9 return $sig$
```

candidates does not exceed this value. The signature of the input MD of a cut serves as an additional label of an exact synthesis query, as evidenced by Theorem 3.

Theorem 3. *Given two n-cuts $\mathcal{C}_1$ and $\mathcal{C}_2$ with identical local functions and distinct input MD $\mathcal{L}_1$ and $\mathcal{L}_2$. If the signatures derived from $\mathcal{L}_1$ and $\mathcal{L}_2$ following Algo. 4 are the same, cuts $\mathcal{C}_1$ and $\mathcal{C}_2$ share the same FHE-cost-optimal circuit implementation.*

Proof. Follows from Lemmata 1 and 2. □

To summarize, effective classification of exact synthesis queries is realized by labelling each query with (1) the NPN representative function of the cut's local function, and (2) the signature of the cut's input MD, preventing meaningless invocation of the paradigm (Algo. 3).

5. MC-aware MD Optimization

Building upon Algo. 3, we introduce an innovative MC-aware MD minimization algorithm. Also, we engineer an FHE circuit optimization flow. Central to this flow is our MC-aware MD minimization algorithm, complemented by the integration of an advanced MD reduction algorithm, ESOP balancing [26].

5.1. *Overview of the Algorithm*

Algo. 5 outlines our MC-aware MD minimization algorithm. It takes a baseline XAG implementation of the target function as input and outputs one with minimized FHE cost.

A cache is utilized to store determined optimal implementations, preventing redundant calls to exact synthesis. As introduced in Section 4.5, NPN-equivalent representative functions and signatures derived from input levels are employed as indices to categorize queries for optimal implementations of cuts effectively.

The algorithm traverses all nodes forming the critical path in topological order (lines 4). For a given node n, cuts rooted at it are enumerated, and their optimal implementations

Algorithm 5: MC-aware MD minimization

```
Input: XAG implementation of the target function, $N$.
Output: Optimized XAG $N_{opt}$.
$N_{opt} \leftarrow N$
$cache \leftarrow$ optimal circuits of small-scale functions
set of cuts $\mathbf{C} \leftarrow cut\_enumeration(N_{opt})$
foreach node $n$ on $N_{opt}$'s critical path in topological order do
    $impl_{new} \leftarrow \emptyset$
    foreach cut $\mathcal{C} \in$ cuts rooted on $n$ $\mathbf{C}[n]$ do
        $f \leftarrow$ local function of cut $\mathcal{C}$
        $\{$representative function $f_r$, signature $sig\} \leftarrow NPN\_classification\ (f, \mathcal{C}.\mathcal{L})$
        $sig \leftarrow derive\_signature(sig)$
        $impl_{new}[\mathcal{C}] \leftarrow cache[\{f_r, sig\}]$
        if $impl_{new}[\mathcal{C}] = NULL$ then
            $impl_{new}[\mathcal{C}] \leftarrow exact\_synthesis\_for\_cuts(\mathcal{C})$
            $cache[\{f_r, sig\}] \leftarrow impl_{new}[\mathcal{C}]$
    cut $\mathcal{C}' \leftarrow \arg\min_{\mathcal{C} \in \mathbf{C}[n]}(\mathcal{C}.\delta_n)$
    rewrite $\mathcal{C}'$ with $impl_{new}[\mathcal{C}']$
return $N_{opt}$
```

are obtained and collected in $impl_{new}$. For a cut $\mathcal{C}$ with a local function f and input MD $\mathcal{L}$, we determine the NPN-equivalent representative function f_r and derive the signature of $\mathcal{L}$, denoted as sig, which are used to access the cache (lines 7-8). If the optimal implementation is not yet in the cache, exact synthesis (Algo. 3) is employed, and the cache is subsequently updated (lines 11-12). Among the optimal implementations of all cuts rooted at node n, the one resulting in the minimum δ_n, i.e., the lowest level of MD at node n, is selected to replace the original implementation of the corresponding cut in the baseline circuit (lines 13-14).

5.2. *An FHE Circuit Optimization Flow*

Intuitively, the proposed MC-aware MD minimization algorithm and ESOP balancing, the most advanced MD optimization algorithm in the literature, explore orthogonal design spaces: In MC-aware MD minimization, the carefully devised AND fence candidate selection scheme excludes those local moves that reduce MD at the cost of a significant increase in MC. In contrast, ESOP balancing aggressively minimizes MD by balancing the ESOP representation of any encountered cut without considering MC. Their distinct action logics inspire us to build a flow that combines the strengths of both.

Our FHE circuit optimization flow is outlined in Algo. 6. The parameter *num_restarts* controls the number of rounds involved in each execution of the flow. In each round, the decision of which optimization algorithm, out of the two candidates, to apply is made by the *random_engine* (line 7). Once one algorithm is recognized to be unable to further optimize the circuit, *switch_opt_algo* intentionally selects the other algorithm for optimization; If neither of the two algorithms can achieve further optimization, the round is terminated (lines 9-15). The best design encountered so far is recorded and is regarded as the starting point for the following rounds of exploration. To break out of local optimal, every restart begins with a *relaxation* (line 5). This procedure significantly

Algorithm 6: FHE Circuit Optimization Flow

```
Input: XAG $N$ that implements the target function.
Output: Optimized XAG $N$.
Parameter : $num\_restarts$, $cost\_metric$.
$N_{best} \leftarrow N$, $N_{opt} \leftarrow N$
for $i \leftarrow 1$ to $num\_restarts$ do
    $N \leftarrow N_{opt}$
    if $i \neq 1$ then
        $N \leftarrow relaxation(N)$
    while $true$ do
        optimization algorithm $\eta \leftarrow random\_engine()$
        $N_{opt} \leftarrow apply\_opt\_algo(N, \eta)$
        if $cost\_metric(N_{opt}) \geq cost\_metric(N)$ then
            $\eta \leftarrow switch\_opt\_algo(\eta)$
            $N_{opt} \leftarrow apply\_opt\_algo(N, \eta)$
            if $cost\_metric(N_{opt}) \geq cost\_metric(N)$ then
                break
        else
            $N \leftarrow N_{opt}$
    if $cost\_metric(N) < cost\_metric(N_{best})$ then
        $N_{best} \leftarrow N$
return $N_{best}$
```

increases the MC and MD of the circuit by representing all XOR nodes in the XAG as a combination of three AND nodes, following $a \oplus b = (\overline{a} \wedge b) \vee (a \wedge \overline{b})$ and $a \vee b = \overline{\overline{a} \wedge \overline{b}}$. We also introduce the parameter *cost_metric* to enable tuning of the cost metric in our optimization flow. This setting facilitates investigating the impact of the adopted cost metric on the quality of the optimized FHE circuit. As an example, we later explore two configurations in the experimental stage: (1) MD, the commonly used cost metric in prior works on FHE circuit optimization, and (2) FHE cost, the cost metric that motivates this work.

6. Experimental Evaluation

In this section, we showcase our experimental results, organized into two main parts:

(1) Evaluation of MC-aware MD minimization as a standalone optimization algorithm.
(2) Assessment of the FHE circuit optimization flow, which integrates MC-aware MD minimization and ESOP balancing, with MD and FHE cost tested as the cost metric.

6.1. *Experimental Setups*

All experiments were conducted on an Apple M1 Max chip with 32GB memory. The following details outline the setups.

Implementation: Our implementation is open-sourced and publicly available at https://github.com/MingfeiYu/hatter. The proposed MC-aware MD minimization algorithm is implemented using the C++ logic network library `mockturtle`. The exact synthesis solver is based on the C++ reasoning library `bill`, with `glucose` [3] chosen as the underlying SAT solver. Both `mockturtle` and `bill` are part of the EPFL logic synthesis libraries [35]. The back-end executor is built upon the homomorphic encryption library `HElib` [25], with the BGV scheme selected and the plaintext space configured to binary. We set the security level to 128-bit(i.e., $\lambda = 128$), and other parameters, within the context of leveled FHE, are configured accordingly by `HElib` to ensure the entire computation can be correctly performed without invoking bootstrapping. To manage the growth of ciphertext size during the computation, we conservatively apply relinearization after each homomorphic AND operation.

Baseline: `LOBSTER` [28], the state-of-the-art FHE circuit optimization tool is chosen as the baseline to compare to. Additionally, we introduce an assessment of ESOP balancing [26], recognized as the most advanced MD reduction algorithm. This evaluation serves as the first exploration of ESOP balancing's performance in the context of FHE circuit optimization. As both ESOP balancing and the proposed MC-aware MD minimization are cut-based, we uniformly set the target cut size as 5 in this experiment.

Benchmark: The 25 involved benchmarks are aligned with [28]. They are collected from four benchmark suites: `Cingulata` benchmarks [9], homomorphic sorting benchmarks [10], *Hacker's Delight* benchmarks [40], and EPFL benchmarks [2]. The functions of the benchmarks vary from medical diagnosis, sorting, and bit-twiddling hacks to random/control logic and are believed to represent computations of interest in potential FHE applications. See [28] for a detailed description of the benchmarks.

6.2. *Evaluating MC-aware MD minimization*

In Table 2, we report MC, MD, the circuit optimization time in seconds (Opt.)[4], and the circuit execution time in seconds (Exec.). Both ESOP balancing and MC-aware MD minimization are iteratively applied until convergence, and the number of runs is recorded (#iter.). The shortest execution time for each benchmark is highlighted in blue. If no improvement over the initial circuit is achieved by any algorithms, no marking is made.

Circuit optimization time: As MC-aware MD minimization relies on on-the-fly exact synthesis to provide the optimal implementations for the encountered cuts, the optimization time is typically longer than ESOP balancing. However, the runtime overhead proves to be acceptable. We attribute this achievement to our well-designed SAT encoding, exact synthesis paradigm, and strategies to avoid unnecessary SAT solving, such as the exact synthesis query classification technique. Due to the unavailability of the source code for `LOBSTER`, we are unable to measure its circuit optimization time directly. However, as reported in [28], `LOBSTER` allocates a *125-hour* time budget for the rewriting rule learning phase for *each benchmark*, and the term rewriting phase takes around *eight hours* for relatively large benchmarks such as msort, isort, and bsort. In comparison, our MC-aware MD minimization achieves circuit optimization four orders of magnitude faster than `LOBSTER`.

Quality of optimized circuits: Among the 25 benchmarks evaluated, improved circuit designs were achieved for 21 benchmarks by at least one of the approaches. Within

[4]Runtime shorter than 0.005s is written as 0.00s

Table 2. Comparing MC-aware MD minimization against state-of-the-art.

Benchmark	Initial			LOBSTER			ESOP Balancing					MC-aware MD Minimization				
	MC	MD	Exec.[s]	MC	MD	Exec.[s]	MC	MD	#iter.	Opt.[s]	Exec.[s]	MC	MD	#iter.	Opt.[s]	Exec.[s]
cardio	109	10	14.47	116	8	10.00	120	8	3	0.11	10.37	108	8	3	9.81	**9.35**
dsort	708	9	59.91	793	8	51.84	948	7	2	0.13	50.49	708	8	2	3.22	**45.18**
msort	810	45	1964.69	1450	36	**1125.30**	1569	36	8	0.75	1229.30	774	45	2	0.74	1228.88
isort	810	45	1830.12	1482	36	**1069.88**	1569	36	8	0.75	1227.46	774	45	2	0.49	1212.08
bsort	810	45	1900.51	1482	36	**1112.19**	1569	36	8	0.75	1226.71	774	45	2	0.49	1213.17
osort	702	25	520.77	1404	20	333.97	1404	20	6	0.47	405.50	638	25	2	0.62	**273.56**
hd01	87	6	5.16	87	6	5.52	102	5	2	0.00	**3.42**	87	6	1	0.27	5.53
hd02	76	6	5.67	76	6	5.24	76	6	1	0.00	5.15	76	6	1	0.27	5.13
hd03	27	5	1.78	27	5	1.38	30	4	2	0.02	1.45	29	4	2	0.89	1.43
hd04	75	10	20.79	78	8	9.20	74	7	3	0.05	**5.09**	63	8	4	9.53	6.03
hd05	121	7	9.15	121	7	8.02	184	6	2	0.01	10.00	121	7	1	0.09	7.95
hd06	121	7	7.11	121	7	8.00	184	6	2	0.02	10.02	121	7	1	0.06	8.00
hd07	17	5	4.25	13	3	0.99	19	3	2	0.01	0.51	15	3	3	0.78	**0.48**
hd08	18	6	2.03	18	5	1.00	26	4	2	0.01	1.24	16	5	2	0.17	**0.94**
hd09	134	14	30.42	177	10	20.73	173	10	4	0.06	17.60	134	10	4	8.78	**14.78**
hd10	35	6	3.54	36	5	1.71	34	5	1	0.01	1.69	32	5	4	1.51	**1.66**
hd11	391	18	133.65	391	15	93.54	411	13	2	0.08	**79.46**	385	14	3	9.74	83.46
hd12	116	16	29.73	116	15	26.53	126	12	3	0.09	**20.25**	107	13	3	0.76	24.22
bar	3141	12	460.61	3015	11	370.77	2266	8	2	0.15	163.12	1841	9	4	28.85	**149.08**
cavlc	655	16	144.26	668	10	63.88	713	8	7	0.37	**52.67**	607	11	7	42.89	76.46
ctrl	107	8	9.87	120	5	4.15	107	4	5	0.03	3.97	89	4	5	9.85	**3.49**
dec	304	3	4.03	304	3	4.08	304	3	1	0.01	4.13	292	3	2	0.32	**3.97**
i2c	1157	15	311.52	1215	8	93.34	1254	7	9	0.24	**71.53**	1152	10	6	19.95	109.30
int2float	213	15	56.27	234	8	17.43	240	7	5	0.10	**14.14**	205	10	5	4.34	21.21
router	170	19	97.48	190	10	30.18	232	9	5	0.13	**19.34**	186	13	4	1.95	37.28
Total			7627.78			4468.87					4634.61					4542.62
Norm.			1.71			1.00					1.04					1.02

these 21 benchmarks, the updated best-known implementations are attributed as follows: LOBSTER accounts for 3, ESOP balancing for 8, and MC-aware MD minimization for 10 of the improved designs. Interestingly, compared to LOBSTER and ESOP balancing, MC-aware MD minimization reaches saturation at significantly different design points. Circuits optimized by MC-aware MD minimization consistently exhibit the lowest MC among the three, aligning with the algorithm's philosophy of considering both MC and MD in the optimization process. Additionally, while both ESOP balancing and LOBSTER focus exclusively on MD reduction, we observed that ESOP balancing consistently achieves a lower, or at least equal, MD compared to LOBSTER.

Discrepancy in the performance of MC-aware MD Minimization: Analysis of cases where the best designs are achieved applying either LOBSTER or ESOP balancing reveals instances where MC-aware MD minimization fails to produce low-FHE-cost designs. This discrepancy can be attributed to the observation that MC-aware MD minimization typically converges with fewer iterations compared to ESOP balancing. In exploring the design space, the strict adherence to slightly increased MC may overlook intermediate design points with significantly higher MC, which can prevent reaching designs with reduced FHE costs. This observation motivates the joint application of MC-aware MD minimization and ESOP balancing, as introduced in Section 5.2. By leveraging ESOP balancing's ability to reduce MD without constraining MC increases, this approach complements MC-aware MD minimization and enables a more comprehensive exploration of the design space.

6.3. *Evaluating FHE Circuit Optimization Flow*

In this experiment, we evaluate the FHE circuit optimization flow devised in Section 5.2. Two different settings of cost metrics are considered and are distinguished as *FHE-cost-oriented* and *MD-oriented* in Table 3. When running the flow, the number of restarts is set to 5 times to strike a balance between the circuit optimization time and the efforts in design space exploration. The fastest execution achieved on each benchmark, with the results shown in Table 2 considered as well, is highlighted in blue. No highlights are applied if the circuits optimized by the flow under both settings are not better than the initial designs or those from previous experiments. When the optimization flow produces optimal designs with identical MC and MD values under both settings, both circuit execution times are highlighted, even if there are slight differences. The total execution time is normalized to that achieved by LOBSTER.

The optimization flow, with either FHE cost or MD configured as the cost metric, offers the best implementations for 17 out of the 21 benchmarks where designs better than the initial ones are produced. More remarkably, 11 designs have never been discovered before by solely applying one of the three optimization algorithms (i.e., LOBSTER, ESOP balancing, and MC-aware MD minimization), evidencing the flow's ability to explore the design space comprehensively. For instance, on benchmarks msort, isort, and bsort, where LOBSTER used to dominate the other two optimization algorithms, with FHE cost configured as the cost metric, the flow achieved implementations with homomorphic execution respectively 21.68%, 14.43%, and 20.84% faster than those optimized by LOBSTER. Notably, in the MD-oriented setting, the flow discovered designs with the new lowest MD values for benchmarks bar, ctrl, and int2float. This also provides convincing

Table 3. Exploring cost metrics within the proposed flow.

Benchmark	FHE-cost-oriented				MD-oriented			
	MC	MD	Opt.[s]	Exec.[s]	MC	MD	Opt.[s]	Exec.[s]
cardio	108	8	70.36	**9.35**	117	8	45.61	10.16
dsort	708	7	26.77	**39.42**	948	7	38.28	50.50
msort	788	42	53.86	**881.38**	1391	36	182.86	1180.46
isort	816	42	45.56	**915.46**	1324	36	88.24	1178.53
bsort	788	42	45.45	**880.41**	1354	36	107.31	1214.01
osort	750	24	20.54	320.54	1261	20	121.63	394.29
hd01	102	5	0.14	**3.42**	102	5	0.22	**3.38**
hd02	76	6	3.53	5.13	76	6	1.89	5.15
hd03	30	4	3.88	1.27	30	4	1.46	1.49
hd04	67	7	17.38	**4.81**	74	7	14.05	5.11
hd05	121	7	7.95	8.08	184	6	5.66	10.03
hd06	121	7	5.18	7.99	184	6	7.06	10.00
hd07	15	3	1.02	**0.48**	15	3	0.65	**0.49**
hd08	21	4	0.67	**0.92**	21	4	0.62	**1.12**
hd09	155	10	14.84	16.38	150	10	11.46	16.00
hd10	32	5	1.44	**1.67**	32	5	0.55	**1.71**
hd11	423	13	32.12	83.88	410	13	22.90	80.88
hd12	115	12	3.73	**19.35**	115	12	3.96	**19.30**
bar	1942	8	127.06	**145.10**	2710	7	185.75	160.52
cavlc	691	9	122.10	56.19	717	8	85.62	53.05
ctrl	97	4	10.76	3.71	115	3	12.94	**2.02**
dec	292	3	1.81	**3.96**	292	3	2.28	**3.94**
i2c	1252	7	57.85	**70.74**	1236	8	22.80	91.04
int2float	217	8	32.24	17.24	309	6	10.49	17.10
router	229	9	19.38	**19.06**	257	9	15.03	22.07
Total				3515.94				4532.35
Norm.				0.79				1.01

evidence that by hybridly applying MC-aware MD minimization and ESOP balancing, the flow can explore the design space that solely applying any existing algorithm failed to reach.

Impact of cost metric selection: When utilizing FHE cost as the cost metric, the optimization flow yields optimal implementations for 15 out of the 21 improved benchmarks, resulting in a 21.32% reduction in homomorphic execution time compared to LOBSTER. Moreover, when compared to the initial circuit designs, the reduction increases to 53.91%, corresponding to a speedup of more than a factor of two. Conversely, when MD is selected as the cost metric, optimal implementations are achieved for only 7 benchmarks, with a total execution time slightly inferior to LOBSTER. It is noteworthy that the MD-oriented setting consistently produces circuits with the lowest MD for nearly all benchmarks, except i2c. Despite its superiority in MD, the resulting designs exhibit inferior execution times, highlighting the importance of carefully managing increases in MC when optimizing FHE circuits.

7. Discussion

This section reflects on the capabilities, limitations, and future potential of the proposed MC-aware MD minimization-centered Boolean FHE circuit optimization framework. While our method achieves notable improvements in homomorphic evaluation performance by simultaneously addressing circuit MC and MD, it also opens new directions for further research. We discuss the significance of our joint optimization strategy, explore how the approach could be extended to arithmetic FHE circuits, and highlight open challenges in cost modeling, particularly regarding the impact of multiplicative levels on runtime.

7.1. *The Importance of Joint MC and MD Optimization*

The experimental results in this work highlight the necessity of jointly optimizing both MC and MD in FHE circuit design. Our analysis demonstrates that reducing MD without regard for increases in MC can lead to suboptimal or even degraded homomorphic evaluation performance. This underscores the need for circuit optimization frameworks that explicitly account for both metrics — an objective achieved by the approach presented here. It should be emphasized that, while we adopted MC $\times$ MD^2 as a practical and empirically grounded cost metric, the main contribution of this work is the development of a reconfigurable, MC-aware MD optimization algorithm. This algorithm allows users to specify custom cost models, supporting future research and application-driven exploration of optimal FHE circuit cost formulations.

7.2. *Extension to Arithmetic FHE Circuits*

While the proposed MC-aware MD minimization technique is demonstrated primarily on Boolean circuits, its applicability extends naturally to arithmetic FHE circuit optimization. Although our exact synthesis framework relies on SAT for functionality verification (see Section 3.2), which is limited to the binary field, this can be generalized. By adopting a *satisfiability modulo theories* (SMT) approach, equivalence checking can be performed over arbitrary finite fields, thus accommodating arithmetic circuits over different moduli. With this extension — while preserving the concept of AND fence enumeration (see Section 3.1.3) — our methodology provides an effective strategy for low-cost arithmetic circuit synthesis in FHE, broadening the impact and flexibility of the proposed approach.

7.3. *Limitations and Directions for Future Work*

A notable limitation of the current study arises from the use of a uniform cost model, which assumes that each homomorphic multiplication incurs the same computational expense, regardless of its multiplicative level. Our experimental results — particularly for the msort, isort, and bsort benchmarks — indicate that circuits optimized by `LOBSTER`, while exhibiting higher FHE cost values due to increased MC, can nonetheless outperform the initial circuit designs in terms of actual execution time. This observation suggests that the cost of homomorphic multiplications may be more accurately modeled as a function of their specific multiplicative level.

As discussed in the introduction, each homomorphic multiplication in a leveled FHE scheme consumes one level of the noise budget. As a ciphertext progresses through successive multiplications, its remaining noise budget (i.e., the number of available modulus prime factors) decreases, resulting in ciphertexts of smaller size and, consequently, faster subsequent homomorphic multiplications. Despite the significance of this effect for FHE circuit optimization, it has yet to be fully incorporated into prevailing cost models.

Nevertheless, this limitation does not diminish the central contribution of this work: to our knowledge, this is the first algorithmic framework to achieve joint MC-MD optimization for FHE circuit synthesis — a significant advance not only within the domain of FHE circuit optimization, but also in the broader context of Boolean circuit optimization. This foundational achievement paves the way for further improvements as cost models and evaluation strategies become increasingly sophisticated.

8. Conclusion

Existing research in homomorphic computation acceleration via circuit optimization has predominantly focused on reducing the MD of Boolean circuits. While MD influences the execution time of individual homomorphic operations, the overall execution time is contingent not only on MD but also on MC. Recognizing this intricate relationship necessitates a more nuanced approach in incorporating the trade-off between MD reduction and MC increase into the FHE circuit optimization problem, to achieve enhanced homomorphic computation acceleration. This study is the first to undertake this challenge.

Based on empirical observations, we adopt FHE cost, formulated as $\mathrm{MC}\times\mathrm{MD}^2$, as a refined metric for FHE circuit optimization, which necessitates the simultaneous optimization of both circuit MC and MD. To address this, we introduce (a) an exact algorithm for synthesizing FHE-cost-optimal circuits, (b) a heuristic algorithm named MC-aware MD minimization that leverages the exact algorithm to efficiently optimize FHE circuit designs, and (c) an FHE circuit optimization flow that integrates our proposed algorithms with existing MD reduction techniques from the literature. Experimental evaluations demonstrate that the optimized FHE circuits achieve an average $1.27\times$ speedup in homomorphic computation, along with substantial reductions in circuit optimization time. These improvements mark a significant step toward realizing lower-latency and lower-power-consumption general-purpose FHE applications.

Simultaneously, this work lays the groundwork for further advancements in FHE circuit optimization. The optimal cost metric for FHE circuit optimization varies depending on the specific FHE scheme and security parameters, a topic that warrants further exploration. This study provides a practical tool to tackle this challenge, as both the proposed MC-aware MD minimization algorithm and the FHE circuit optimization flow can be readily adjusted to accommodate different specifications of FHE cost.

Funding Open access funding provided by EPFL Lausanne

A. Boolean Classification Techniques

Boolean function classification is the process of categorizing Boolean functions into different classes based on various characteristics and properties of the functions. The two Boolean classification techniques employed in this paper are *NPN classification* and *affine function classification*.

A.1. *NPN Classification*

Based on an n-variable Boolean function $f(x_1, \cdots, x_i, \cdots, x_j, \cdots, x_n)$, the three NPN-equivalent operations are:

1. Input negation: $f \xrightarrow{x_i \to \overline{x_i}} f'$.
2. Input permutation: $f \xrightarrow{x_i \leftrightarrow x_j} f'$.
3. Output negation: $f \xrightarrow{\overline{f}} f'$.

Theorem 4. *NPN-equivalent operations are MD-preserving.*

Proof. Without loss of generality, we consider the impact of input permutation on MD, as it is evident that input and output negations do not affect the MD of an XAG. Recall that, as revealed by Eq. 5, the MD d of an XAG with n PIs satisfies

$$d = \max\{\sigma_a + \Delta_a \mid 1 \leq a \leq n\}.$$

If we permute any two input variables, say x_i and x_j, the values of σ_i and σ_j are exchanged, as are the values of Δ_i and Δ_j. However, this exchange does not affect the value of d. Therefore, NPN-equivalent operations are MD-preserving. □

A.2. *Affine Function Classification*

Affine function classification is a more effective Boolean function classification technique, in the sense that the set of affine-equivalent operations is the superset of the set of NPN-equivalent operations. Besides the three NPN-equivalent operations, affine-equivalent operations further include:

1. Translational operation: $f \xrightarrow{x_i \to (x_i \oplus x_j)} f'$.
2. Disjoint translational operation: $f \xrightarrow{\oplus x_i} f'$.

Due to the inclusion of the translational operation, affine-equivalent operations are conditionally MD-preserving, as demonstrated in the following sub-section.

A.3. *Proof of Theorem 1*

Proof. Noting that input negation, input permutation, and output negation have been demonstrated to be MD-preserving operations, without loss of generality, we focus our analysis exclusively on the impact of translational operations and disjoint translational operations on MD.
Recall that the MD d of an XAG with n PIs satisfies

$$d = \max\{\sigma_a + \Delta_a \mid 1 \le a \le n\}.$$

Considering a translational operation applied to two input variables, x_i and x_j, the term $\sigma_i + \Delta_i$ is transformed into $\max\{\sigma_i, \sigma_j\} + \Delta_i$. The value of d may change only if:

$$\sigma_j > \sigma_i, \text{and } \sigma_j + \Delta_i > \max\{\sigma_a + \Delta_a \mid 1 \le a \le n\}.$$

However, when the input variables share an identical MD ($\sigma_i = \sigma_j$), the aforementioned condition is excluded, demonstrating the MD-preserving nature of translational operations.
For a disjoint translational operation on any input variable x_i, a direct path from x_i to the PO of the XAG is introduced. This operation does not affect the critical paths in the XAG, and consequently, it does not alter the value of d.
Thus, we conclude that affine-equivalent operations are MD-preserving if the input variables share an identical MD. □

B. Proof of Theorem 2

Proof. We prove the contrapositive. Assume there exists an abstract XAG N_2 that implements f with fence $\mathcal{F}_2 = (c_1, \ldots, c_{d-1})$ of MD $d-1$. We construct an abstract XAG N_1 of MD d that still implements f.
Let r denote the (unique) PO signal of N_2. Form N_1 by *adding one final step* at local MD level d whose two XOR-cloud fan-ins are both the singleton $\{r\}$, i.e., the new step computes

$$r' = r \wedge r = r.$$

This preserves the overall Boolean function (since $r \wedge r \equiv r$), increases the depth by exactly one, and respects the abstract-XAG constraints: each fan-in comes from a strictly lower MD level and at least one comes from the immediately preceding level. The resulting fence is $\mathcal{F}_1 = (c_1, \ldots, c_{d-1}, 1)$, which has MD d.
Thus, if a realization exists at depth $d-1$, a realization also exists at depth d. Taking the contrapositive, if no abstract XAG exists at MD d, then none exists at MD $d-1$. □

C. Proof of Signature Derivation Rules

C.1. *Proof of Lemma 1*

Proof. Fix any fence $\mathcal{F}$ of depth d and any schedule $\mathcal{S} = (\Delta_1, \ldots, \Delta_n) \in \{0, \ldots, d\}^n$ (the earliest local-MD level at which each leaf may be used). For an input-MD vector $\mathcal{L}$, the root MD realized by $(\mathcal{F}, \mathcal{S})$ equals

$$\sigma_{\text{root}}(\mathcal{F}, \mathcal{S}; \mathcal{L}) = \max_{1 \le i \le n} \{\sigma_i + \Delta_i\}.$$

If $\sigma_i' = \sigma_i - a$ for all i, then

$$\sigma_{\text{root}}(\mathcal{F}, \mathcal{S}; \mathcal{L}_2) = \max_i\{\sigma_i' + \Delta_i\} = \max_i\{\sigma_i + \Delta_i - a\} = \sigma_{\text{root}}(\mathcal{F}, \mathcal{S}; \mathcal{L}_1) - a.$$

Hence, for any two candidate implementations N_1 and N_2 (each induced by some $(\mathcal{F}_1, \mathcal{S}_1)$ and $(\mathcal{F}_2, \mathcal{S}_2)$), the ordering of their root MDs is preserved under the shift from $\mathcal{L}_1$ to $\mathcal{L}_2$:

$$\sigma_{\text{root}}(N_1; \mathcal{L}_1) \le \sigma_{\text{root}}(N_2; \mathcal{L}_1) \iff \sigma_{\text{root}}(N_1; \mathcal{L}_2) \le \sigma_{\text{root}}(N_2; \mathcal{L}_2).$$

Therefore, the set of minimizers (optimal implementations) is the same for $\mathcal{L}_1$ and $\mathcal{L}_2$. □

C.2. *Proof of Lemma 2*

Proof. Let d be the minimum root MD achievable for $\mathcal{C}_1$. Consider any candidate implementation (fence and schedule), and let Δ_a be the maximum number of AND levels from leaf x_a to the root (so $0 \le \Delta_a \le d$). The root MD under $\mathcal{L}_1$ is

$$\sigma_{\text{root}}(\mathcal{L}_1) = \max_{1 \le a \le n} \{\sigma_a + \Delta_a\}.$$

Step 1 (non-critical prefix). Because $\sigma_i - \sigma_{i-1} \ge d$ and $\Delta_a \le d$, for every $a \le i-1$ we have

$$\sigma_a + \Delta_a \le \sigma_{i-1} + d \le \sigma_i.$$

Hence, leaves $1, \dots, i-1$ can never exceed the contribution of leaf i (whose term is at least σ_i). Therefore,

$$\sigma_{\text{root}}(\mathcal{L}_1) \;=\; \max_{a \ge i}\{\sigma_a + \Delta_a\}.$$

Intuitively, a sufficiently large gap at position i prevents earlier leaves from becoming critical in any depth-d implementation.
Step 2 (flatten the prefix). Define a modified input-MD vector

$$\mathcal{L}_3 = (\underbrace{\sigma_i, \dots, \sigma_i,}_{i \text{ entries}}\ \sigma_{i+1}, \dots, \sigma_n).$$

For any implementation (same Δ),

$$\sigma_{\text{root}}(\mathcal{L}_3) = \max\left\{\sigma_i, \max_{a \ge i}\{\sigma_a + \Delta_a\}\right\} = \max_{a \ge i}\{\sigma_a + \Delta_a\} = \sigma_{\text{root}}(\mathcal{L}_1).$$

Thus, *the ordering of implementations* (and hence, the set of optimizers) is identical under $\mathcal{L}_1$ and $\mathcal{L}_3$.
Step 3 (align to $\mathcal{L}_2$). Observe that $\mathcal{L}_2$ is obtained from $\mathcal{L}_3$ by subtracting the constant σ_i from every entry: the first i entries become 0, and the remaining entries become $\sigma_a - \sigma_i$. By Lemma 1 (Alignment), subtracting a constant from all inputs preserves the order of all candidate implementations (it shifts all $\sigma_a + \Delta_a$ by the same constant). Therefore, the set of optimizers under $\mathcal{L}_3$ and $\mathcal{L}_2$ coincides.
Combining the above steps shows that the optimal implementation for $\mathcal{C}_1$ (under $\mathcal{L}_1$) is also optimal for $\mathcal{C}_2$ (under $\mathcal{L}_2$). □

References

[1] Pascal Aubry, Sergiu Carpov, and Renaud Sirdey. Faster homomorphic encryption is not enough: Improved heuristic for multiplicative depth minimization of boolean circuits. In *Proceedings of the Cryptographers' Track at the RSA Conference*, volume 12006, pp. 345–363. Springer, 2020.

[2] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. The EPFL combinational benchmark suite. In *Proceedings of International Workshop on Logic & Synthesis*, 2015.

[3] Gilles Audemard and Laurent Simon. Glucose SAT solver 4.1, 2017.

[4] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. In *Innovations in Theoretical Computer Science*, pp. 309–325. ACM, 2012.

[5] Robert K. Brayton and Alan Mishchenko. ABC: an academic industrial-strength verification tool. In *Proceedings of the 22nd International Conference on Computer Aided Verification*, volume 6174, pages 24–40. Springer, 2010.

[6] Joan Boyar, René Peralta, and Denis Pochuev. On the multiplicative complexity of boolean functions over the basis ($\wedge$, $\oplus$, 1). *Theor. Comput. Sci.*, 235(1):43–57, 2000.

[7] Sergiu Carpov. A fast heuristic for mapping boolean circuits to functional bootstrapping. Cryptology ePrint Archive, Paper 2024/1204, 2024.

[8] Sergiu Carpov, Pascal Aubry, and Renaud Sirdey. A multi-start heuristic for multiplicative depth minimization of boolean circuits. In *Proceedings of the 28th International Workshop on Combinatorial Algorithms*, volume 10765, pages 275–286. Springer, 2017.

[9] Sergiu Carpov, Paul Dubrulle, and Renaud Sirdey. Armadillo: A compilation chain for privacy preserving applications. In *Proceedings of the 3rd International Workshop on Security in Cloud Computing*, pp. 13–19. ACM, 2015.

[10] Gizem S. Çetin, Yarkin Doröz, Berk Sunar, and Erkay Savas. Depth optimized efficient homomorphic sorting. In *Proceedings of the 4th International Conference on Cryptology and Information Security in Latin America*, volume 9230, pages 61–80. Springer, 2015.

[11] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33:34–91, 2020.

[12] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology – ASIACRYPT 2017*, pp. 409–437, 2017.

[13] Seonyoung Cheon, Yongwoo Lee, Dongkwan Kim, Ju Min Lee, Sunchul Jung, Taekyung Kim, Dongyoon Lee, and Hanjun Kim. Dacapo: Automatic bootstrapping management for efficient fully homomorphic encryption. In *USENIX Security*, 2024.

[14] Sergiu Carpov, Thanh-Hai Nguyen, Renaud Sirdey, Gianpiero Costantino, and Fabio Martinelli. Practical privacy-preserving medical diagnosis using homomorphic encryption. In *Proceedings of the 9th IEEE International Conference on Cloud Computing*, pp. 593–599. IEEE Computer Society, 2016.

[15] Ana Costache and Nigel P. Smart. Which ring based somewhat homomorphic encryption scheme is best? In *Proceedings of the Cryptographers' Track at the RSA Conference*, volume 9610, pages 325–340. Springer, 2016.

[16] Çagdas Çalik, Meltem Sönmez Turan, and René Peralta. The multiplicative complexity of 6-variable boolean functions. *Cryptogr. Commun.*, 11(1):93–107, 2019.

[17] Léo Ducas and Daniele Micciancio. FHEW: Bootstrapping homomorphic encryption in less than a second. In *EUROCRYPT*, pages 617–640, 2015.

[18] Roshan Dathathri, Olli Saarikivi, Hao Chen, Kim Laine, Kristin E. Lauter, Saeed Maleki, Madanlal Musuvathi, and Todd Mytkowicz. CHET: An optimizing compiler for fully-homomorphic neural-network inferencing. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pp. 142–156. ACM, 2019.

[19] Colin R. Edwards. The application of the rademacher-walsh transform to boolean function classification and threshold logic synthesis. *IEEE Trans. Computers*, 24(1):48–62, 1975.

[20] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptol. ePrint Arch.*, pp. 144, 2012.

[21] Craig Gentry. Fully homomorphic encryption using ideal lattices. In Michael Mitzenmacher, editor, *41st ACM STOC*, pp. 169–178. ACM Press, May / June 2009.

[22] Charles Gouert, Dimitris Mouris, and Nektarios Georgios Tsoutsos. SoK: New insights into fully homomorphic encryption libraries via standardized benchmarks. *Proceedings on Privacy Enhancing Technologies*, (3):154–172, 2023.

[23] Zhenyu Guan, Ran Mao, Qianyun Zhang, Zhou Zhang, Zian Zhao, and Song Bian. AutoHog: Automating homomorphic gate design for large-scale logic circuit evaluation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 43(7):1971–1983, 2024.

[24] Eiichi Goto and H. Takahasi. Some theorems useful in threshold logic for enumerating boolean functions. In *Proceedings of the 2nd International Federation for Information Processing Congress*, pp. 747–752, 1962.

[25] Shai Halevi and Victor Shoup. Design and implementation of helib: a homomorphic encryption library. *IACR Cryptol. ePrint Arch.*, pp. 1481, 2020.

[26] Thomas Häner and Mathias Soeken. Lowering the t-depth of quantum circuits via logic network optimization. *ACM Trans. Quantum Computing*, 3(2), 2022.

[27] Donald E. Knuth. *The Art of Computer Programming, Volume 4, Fascicle 4: Generating All Trees–History of Combinatorial Generation*. Addison-Wesley Professional, 2013.

[28] Dongkwon Lee, Woosuk Lee, Hakjoo Oh, and Kwangkeun Yi. Optimizing homomorphic evaluation circuits by program synthesis and term rewriting. In *Proceedings of the 41st ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pp. 503–518. ACM, 2020.
[29] Tancrède Lepoint and Pascal Paillier. On the minimal number of bootstrappings in homomorphic circuits. In *Financial Cryptography and Data Security*, volume 7862, pages 189–200. Springer, 2013.
[30] Alan Mishchenko, Satrajit Chatterjee, and Robert K. Brayton. Improvements to technology mapping for lut-based fpgas. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 26(2):240–253, 2007.
[31] Johannes Mono, Chiara Marcolla, Georg Land, Tim Güneysu, and Najwa Aaraj. Finding and evaluating parameters for BGV. In *Progress in Cryptology - AFRICACRYPT 2023*, pages 370–394, 2023.
[32] Marie Paindavoine and Bastien Vialla. Minimizing the number of bootstrappings in fully homomorphic encryption. In *Proceedings of the 22nd International Conference on Selected Areas in Cryptography*, volume 9566, pages 25–43. Springer, 2015.
[33] Ronald L. Rivest, Len Adleman, and Michael L. Dertouzos. On data banks and privacy homomorphisms. *Foundations of Secure Computation*, 4(11):169–180, 1978.
[34] Mathias Soeken. Determining the multiplicative complexity of boolean functions using SAT, 2020.
[35] Mathias Soeken, Heinz Riener, Winston Haaswijk, and Giovanni De Micheli. The EPFL logic synthesis libraries, 2018.
[36] Meltem Sönmez Turan and René Peralta. The multiplicative complexity of boolean functions on four and five variables. *IACR Cryptol. ePrint Arch.*, pp. 848, 2015.
[37] Eleonora Testa, Mathias Soeken, Luca G. Amarù, and Giovanni De Micheli. Reducing the multiplicative complexity in logic networks for cryptography and security applications. In *Proceedings of the 56th Annual Design Automation Conference*, pp. 1–6. ACM, 2019.
[38] Grigori S. Tseitin. *On the Complexity of Derivation in Propositional Calculus*, pp. 466–483. Springer, 1983.
[39] Jelle Vos, Mauro Conti, and Zekeriya Erkin. Oraqle: A depth-aware secure computation compiler. In *Proceedings of the 12th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pp. 43–50, 2024.
[40] Henry S. Warren. *Hacker's Delight, Second Edition*. Pearson Education, 2013.
[41] Mingfei Yu, Sergiu Carpov, Alessandro Tempia Calvino, and Giovanni De Micheli. On the synthesis of high-performance homomorphic Boolean circuits. In *Proceedings of the 12th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pp. 51–63, 2024.
[42] Mingfei Yu and Giovanni De Micheli. Striving for both quality and speed: Logic synthesis for practical garbled circuits. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design*, pp. 1–9. IEEE, 2023.