On the Synthesis of High-performance Homomorphic Boolean Circuits

Mingfei Yu
Integrated Systems Laboratory, EPFL
Lausanne, Switzerland
mingfei.yu@epfl.ch

Sergiu Carpov
Arcium
Zug, Switzerland
sergiu@arcium.com

Alessandro Tempia Calvino
Integrated Systems Laboratory, EPFL
Lausanne, Switzerland
alessandro.tempiacalvino@epfl.ch

Giovanni De Micheli
Integrated Systems Laboratory, EPFL
Lausanne, Switzerland
giovanni.demicheli@epfl.ch

Abstract

The rapid growth of cloud computing has intensified the need for secure data outsourcing solutions. *Fully homomorphic encryption* (FHE) offers a promising approach by enabling computations on encrypted data without exposing the plaintext. This paper advances the synthesis of homomorphic Boolean circuits, a pivotal component in FHE, by addressing two major limitations of existing methods: (1) effectively exploiting large-fan-in logic gates to achieve more compact circuit designs, and (2) leveraging the *multi-value functional bootstrapping* technique to further reduce computational overhead in evaluating Boolean functions. The proposed homomorphic circuit synthesis flow effectively and efficiently improves homomorphic evaluation efficiency. Experimental results show that, compared to the state-of-the-art, our flow achieves 1.36× faster synthesis time and reduces execution cost by an average of 29.94%, with a maximum reduction of 76.02%.

CCS Concepts

• Theory of computation → Circuit complexity; **• Software and its engineering →** *Compilers.*

Keywords

Homomorphic Encryption; Logic Synthesis; Technology Mapping

ACM Reference Format:
Mingfei Yu, Sergiu Carpov, Alessandro Tempia Calvino, and Giovanni De Micheli. 2024. On the Synthesis of High-performance Homomorphic Boolean Circuits. In *Proceedings of the 12th Workshop on Encrypted Computing & Applied Homomorphic Cryptography (WAHC '24), October 14–18, 2024, Salt Lake City, UT, USA.*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3689945.3694803

1 Introduction

The proliferation of sensitive data in the digital era necessitates robust solutions for secure data processing and storage. *Fully homomorphic encryption* (FHE) has emerged as a transformative technology, enabling computations on encrypted data while preserving privacy. This capability is crucial for applications in cloud computing, where data confidentiality must be maintained even during processing by untrusted third parties.

Modern FHE schemes use noisy ciphertexts, and since noise accumulates with each homomorphic computation, an operation to reduce this noise is necessary. This operation is widely known as *bootstrapping*. While Craig Gentry proposed the first realization of bootstrapping in 2009 [12], making FHE theoretically achievable, it was initially too computationally intensive and impractical. After decades of effort to improve practicality, *fast bootstrapping* FHE schemes have emerged as an ideal solution for evaluating Boolean functions homomorphically. As the name suggests, bootstrapping in these schemes is highly efficient and can evaluate a function while simultaneously reducing noise. For this reason, the bootstrapping operation in fast bootstrapping schemes is specifically termed *functional bootstrapping* (FBS). The most advanced scheme in this line, *torus FHE* (TFHE), can perform an FBS operation in less than 10 milliseconds [8]. In contrast, each bootstrapping operation in *leveled homomorphic encryption* (LHE) schemes, the other main branch of modern FHE schemes, usually takes minutes. However, each encrypted value in fast bootstrapping schemes is typically limited to a binary number or a small integer. This explains why fast bootstrapping FHE schemes are ideal for the homomorphic evaluation of Boolean functions. In this paper, the terms *homomorphic Boolean circuit* and TFHE *circuit* are used interchangeably.

In recent years, we have witnessed many remarkable TFHE compilers that translate Boolean functions, described in high-level programs such as *hardware description language* (HDL) as adopted by `Cingulata` [6], `Romeo` [15], or C++ as adopted by Google's `Transpiler` [14], into homomorphic Boolean circuits, with each gate bound to certain homomorphic computation implemented in the `TFHE`[1] library. These compilers enable non-experts to develop secure and efficient FHE applications, significantly facilitating the transition of recent developments in FHE schemes from theory to

[1]To avoid confusion, we distinguish between the TFHE scheme and the `TFHE` library using different fonts.

practice. However, we observe that the exploration of circuit optimization — an essential task in a compilation process that strongly affects the resulting homomorphic evaluation efficiency — remains insufficient in these compiler designs.

The execution of each homomorphic Boolean gate involves performing an FBS operation, which remains the most computationally intensive task in TFHE. Consequently, the efficiency of homomorphically evaluating a Boolean function is closely correlated with the gate count of the circuit design. This correlation frames TFHE circuit synthesis as a general area-oriented Boolean circuit synthesis problem. It motivates existing TFHE compilers to rely on off-the-shelf *electronic design automation* (EDA) tools, originally developed for hardware circuit synthesis, to achieve compact TFHE circuit designs. For instance, `Romeo` leverages the open-source technology mapper `Yosys` [25] to map HDL designs to two-input Boolean gates and three-input multiplexers supported by the `TFHE` library [15]. A similar methodology is also adopted by `Transpiler` [14]. However, this method has two major limitations. First, it is constrained by the gate set supported by the `TFHE` library. Existing literature indicates that larger fan-in Boolean gates can also be exploited to deliver more compact homomorphic Boolean circuit designs [2, 18], yet effectively exploiting them remains an open problem. Second, the TFHE circuit synthesis problem has unique features not shared by hardware synthesis problems, necessitating the development of a customized circuit synthesis approach to achieve high-performance TFHE circuit designs. Specifically, the *multi-value FBS* technique [7] suggests that, for multiple Boolean gates with the same supports (*i.e.*, inputs), their homomorphic evaluation can share a single FBS operation. From a circuit synthesis perspective, this implies that the cost of a multi-output Boolean gate can be close to that of a single-output gate, presenting an opportunity to synthesize higher-performance TFHE circuits. However, this opportunity has not been thoroughly investigated.

This paper seeks to address these limitations. We approach the synthesis problem as a technology mapping issue. The approach is not limited to a subset of functions and applies to any Boolean function. To that end, we first analyze the intrinsic properties of large-fan-in Boolean gates to curate a suitable gate set for homomorphic Boolean circuit synthesis (Section 3). We then customize a multi-value-FBS-aware and area-oriented mapping algorithm to maximally enhance circuit quality and ultimately reduce the computational overhead of homomorphic evaluation (Section 4). The effectiveness and efficiency of our approach are evidenced by comprehensive experimental evaluation (Section 5). Compared to the state-of-the-art TFHE circuit synthesis approach, our method achieved an average 26.27% reduction in circuit synthesis time and an average 29.94%, up to 76.02%, improvement in homomorphic evaluation efficiency. These contributions apply to other fast bootstrapping schemes as well, such as *FHEW* [10].

2 Background

This section provides an overview of modern FHE schemes (Section 2.1), the preliminaries on TFHE (Section 2.2) and Boolean circuits (Section 2.3), and a summary of existing research on homomorphic Boolean circuit synthesis (Section 2.4).

2.1 Fully Homomorphic Encryption

FHE is a form of encryption that allows computations to be performed on encrypted data without requiring access to the plaintext. This capability is crucial for maintaining data privacy in various applications, such as cloud computing. The concept of FHE was first proposed by Rivest *et al.* in 1978 [21], but it was not until Craig Gentry's breakthrough in 2009 that the first FHE scheme was introduced [12]. Gentry's scheme employed *lattice-based cryptography*, the security of which relies on the *(ring) learning with errors* ((R)LWE) problem, where *noise* is exploited to provide security. However, noise in ciphertexts accumulates with homomorphic operations, leading to failed decryption if it exceeds a certain threshold. To handle the noise growth inherent in homomorphic operations, Gentry introduced the concept of *bootstrapping*, an operation that refreshes the noise level of ciphertexts. While bootstrapping makes FHE theoretically achievable, it is computationally intensive and impractical in its original form.

Modern FHE schemes diverge into two mainstream branches: *leveled homomorphic encryption* (LHE) and *fast bootstrapping*. LHE schemes, such as *BGV* [3] and *BFV* [11], support a predefined number of homomorphic operations on ciphertexts without invoking bootstrapping. In contrast, the bootstrapping operation in fast bootstrapping schemes is realized significantly differently, commonly referred to as *functional bootstrapping* (FBS). An FBS operation not only refreshes ciphertexts but also evaluates an operation simultaneously. The most recent advancement in the fast bootstrapping branch is the *torus FHE* (TFHE) scheme [8], a successor of the *GSW* scheme [13] and the *FHEW* scheme [10], which provides efficient evaluation of Boolean functions.

2.2 Torus FHE

TFHE is a state-of-the-art FHE scheme optimized for fast and efficient evaluation of Boolean functions. TFHE achieves this through an innovative bootstrapping mechanism that maintains low noise levels and high performance. Messages are encrypted as `LWE` samples or ciphertexts, one message per sample. With the plaintext space size denoted as p, the `LWE` encryption of message $m \in \mathbb{Z}_p$ is a tuple $(\mathbf{a}, b)$ such that $b - \mathbf{a} \cdot \mathbf{s} \approx m\frac{1}{p}$.

2.2.1 *Functional bootstrapping.*

The FBS operation in fast bootstrapping schemes (TFHE, FHEW, *etc.*) is a procedure which can evaluate any function $F : \mathbb{Z}_p \to \mathbb{Z}_p$ in addition to ciphertext noise reduction. Input and output of FBS are `LWE` samples. FBS allows to evaluate arithmetic circuits over encrypted data where each operation output ciphertext is bootstrapped. The bootstrapping procedure is implemented using a homomorphic accumulator (encoded in auxiliary `GSW` and `RLWE` LHE schemes) which evaluates the linear part of the decryption function followed by the non-linear part. For this line of schemes, the bootstrapping procedure follows a common pattern and can be split into four steps:

(1) Input `LWE` sample $(\mathbf{a}, b)$ modulus is switched from the torus $\mathbb{T}$ to $\mathbb{Z}_t$. The obtained `LWE` sample is an approximation of the input one. A cyclic multiplicative group $\mathcal{G}$, where $\mathbb{Z}_t \simeq \mathcal{G}$, is used for an equivalent representation of $\mathbb{Z}_t$ elements. $\mathcal{G}$ contains all the powers of $X^k \mod \Phi(X)$ where $\Phi(X)$ is the quotient polynomial defining the `RLWE` scheme. Here t is the smallest

integer verifying $X^t \mod \Phi(X) = 1$. Power-of-two cyclotomic polynomials $\Phi(X)$ is commonly adopted, such as in TFHE.

(2) The phase φ of the modulus-switched LWE ciphertext is transformed to an encryption of X^φ in the auxiliary RLWE scheme. The message $\varphi \in \mathbb{Z}_t$ is obtained from $(\mathbf{a}, b)$ using linear transformation $b - \mathbf{a} \cdot \mathbf{s} \approx \varphi$ (*i.e.*, the linear part of the decryption algorithm) given the GSW encryptions of X^{s_i}, the bootstrapping key, and the so-called *blind rotations* of RLWE ciphertexts. We obtain the so-called *accumulator* ACC which contains an encryption of $X^\varphi \in \mathcal{G}$.

(3) A so-called *test polynomial* (or *test vector*) TV_F is multiplied to ACC. The test polynomial encodes output values of a function F for each possible input message $\varphi \in \mathbb{Z}_t$. The phase φ encoded in $(\mathbf{a}, b)$ is a noised version of the actual message m. Function $F = f \circ r_p$ encodes the composition of the rounding function $r_p : \mathbb{Z}_t \to \mathbb{Z}_p$ and a "payload" function $f : \mathbb{Z}_p \to \mathbb{Z}_p$. The rounding function r_p corresponds to the final non-linear step of LWE decryption.

(4) Finally, the output LWE encryption $f(m) = F(\varphi)$ is extracted from the RLWE encryption $\mathsf{TV}_F \cdot \mathsf{ACC}$.

Fig. 1 illustrates the encoding of plaintext message space $\mathbb{Z}_4$ on the torus and a test vector encoding of the bootstrapped function F. The message space extends beyond $\mathbb{Z}_4$, and in this case, the negated function F values are returned by the FBS. This behavior is due to the *negacyclic* property of the cyclotomic quotient ring, which is a power-of-two cyclotomic polynomial in TFHE. A function $F : \mathbb{Z}_{2p} \to \mathbb{Z}_p$ is negacyclic if it satisfies $F(x) \equiv -F(x + p)$ for any $x \in \mathbb{Z}_p$. In summary, when parameterized with a message space $\mathbb{Z}_p$, an FBS operation can evaluate any function $F : \mathbb{Z}_p \to \mathbb{Z}_p$ or any negacyclic function $F : \mathbb{Z}_{2p} \to \mathbb{Z}_p$.

When homomorphically evaluating an n-input single-output Boolean gate that implements a Boolean function $f : \mathbb{B}^n \to \mathbb{B}$, the n input ciphertexts are first linearly combined into a single ciphertext before applying the FBS procedure described above. This *linear combination* consists of homomorphic additions and scalar multiplications. These simple homomorphic arithmetic operations adjust the scale and offset of the resulting ciphertext with a mildly increased noise. The resulting ciphertext encodes a plaintext value which distinguishes the entries of the Boolean gate's truth table that have different output values. Specifically, the FBS operations for evaluating a two-input one or a three-input multiplexer are termed *gate bootstrapping* operations in the TFHE scheme. The implementation of FBS operations leverages execution-speed-optimized modular arithmetic and polynomial operations to achieve high performance — a gate bootstrapping operation performs in 10 milliseconds [8].

2.2.2 *Multi-value functional bootstrapping.*

Multi-value FBS is a significant advancement in enhancing the efficiency of FBS operations [7]. This technique enables the evaluation of multiple functions over the same encrypted input (*i.e.*, a multi-output Boolean gate) in a single bootstrapping operation, offering substantial improvements in computational efficiency.

Traditionally, evaluating different functions, even on the same encrypted data, requires multiple FBS operations, each corresponding to a different function. Multi-value FBS consolidates these operations by sharing computational steps among different functions. This is at the core of factoring the test polynomials that define the evaluated functions into a common factor and specific parts that correspond to different functions. This allows the heavy computation in a blind rotation to be shared among multiple functions. Indeed, the blind rotation is applied once using the common factor of the test polynomials. After that, the ciphertexts corresponding to different functions are obtained by multiplying the blind-rotated common accumulator with each specific part, which is computationally cheaper as compared to performing multiple blind rotations.

By optimizing the steps involving polynomial multiplications and using low-norm test polynomials, the technique effectively manages the noise in the resulting ciphertexts, maintaining the correctness of homomorphic evaluation. The benchmarks from [7] report an execution time of 1.6 seconds for a 6-input m-output Boolean gate. Notably, the execution time is almost independent of the output count m. The authors did not provide benchmarks for Boolean gates with smaller fan-in sizes, but it is reasonable to assume that the execution time would decrease super-linearly with a decrease in fan-in size.

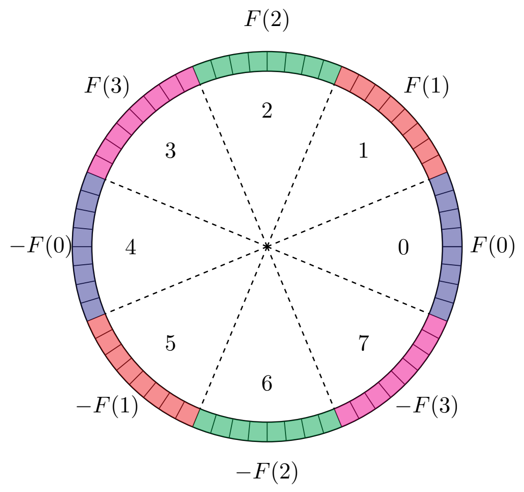


Figure 1: Example of TFHE message space $\mathbb{Z}_4$ (separated by dashed lines) and FBS function F encoding (colored segments).

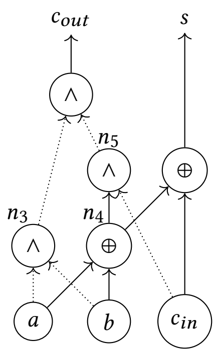


Figure 2: An XAG implementation of a 1-bit full adder.

2.3 Boolean Circuit

2.3.1 Basic concepts. A Boolean circuit, or a logic network, is a *directed acyclic graph* (DAG) $G = (V, E)$, where V is the set of nodes and E is the set of edges. Each edge in E is directed and models a wire from node n_i to node n_o, where $n_i, n_o \in V$; We term n_i as a *fan-in* of n_o and n_o as a *fan-out* of n_i. The presence of an inverter is recorded as an attribute of the edges. A Boolean circuit is further characterized by its *primary inputs* (PIs) and *primary outputs* (POs), denoted as set I and set O, respectively. Each node in V is either a PI or a logic gate, *i.e.,* $I \subseteq V$. Each PO in O can be either a PI or a logic gate and conceptually models an outgoing wire, with potential complementation, from this node. Fig. 2 showcases a logic network that implements a one-bit full adder. More specifically, the network can be termed as an *XOR-AND-inverter graph* (XAG), as the logic primitives are constrained to two-input AND nodes (denoted as '$\wedge$') and two-input XOR nodes (denoted as '$\oplus$'). A dotted edge indicates an inversion. The XAG consists of eight nodes, where $I = \{a, b, c_{in}\}$ and $O = \{c_{out}, s\}$.

2.3.2 Cuts. A *cut* is a commonly used concept to highlight a part of a logic network. A valid cut C features a root node r and a set of leaf nodes L, such that: (1) Any path from a PI to the root r includes at least one leaf node $l \in L$; and (2) Each leaf l is on at least one such path. Intuitively, the part of the network highlighted by a cut, called a *logic cone*, is the set of all nodes on any path from a leaf node to the root node. We call a cut k-feasible if its number of leaf nodes does not exceed k. For instance, in the XAG in Fig. 2, there are three non-trivial 3-feasible cuts rooted at node c_{out}, denoted as C_0 to C_2, with leaves $\{n_3, n_5\}$, $\{n_3, n_4, c_{in}\}$, and $\{a, b, c_{in}\}$, respectively. The process of finding the k-feasible cuts rooted at all non-PI nodes in a logic network is known as *cut enumeration* [9].

We define the *global function* of node n, f_n, as the Boolean function implemented by n in terms of the PIs, *i.e.,* $f_n{:}\mathbb{B}^{|I|} \rightarrow \mathbb{B}$. In contrast, we define the Boolean function implemented by cut C, whose root node is n and leaf nodes are L, as a *local function* of node n in terms of C, $f_n^C{:}\mathbb{B}^{|L|} \rightarrow \mathbb{B}$. Reusing the example in Fig. 2, we have

$$f_{c_{out}}^{C_0} = \neg n_3 \wedge \neg n_5,$$

where '$\neg$' denotes negation. Since the leaf nodes of cut C_2 are exactly the PIs, we have

$$f_{c_{out}}^{C_2} = f_{c_{out}} = \langle a, b, c_{in} \rangle,$$

where '$\langle\rangle$' denotes the Boolean majority operation.

2.3.3 Maximum fanout-free cone (MFFC). The *MFFC* of a node n refers to the set of nodes in the fan-in cone of n such that any path from these nodes to any PO passes through n. For example, in Fig. 2, the MFFC of node c_{out} consists of node n_3 and n_5. Conversely, there are no nodes in the MFFC of node s, as its two fan-in nodes, node n_4 and node c_{in}, also contribute to node n_5. Intuitively, the nodes in n's MFFC exclusively contribute to n and do not affect other parts of the logic network. Therefore, measuring the size of the MFFC of a node provides a local estimation of the area cost associated with it — if this node is removed from the circuit, the nodes in its MFFC can also be eliminated.

2.4 Related Works

We review related works on the two open problems in homomorphic Boolean circuit synthesis, to which this work contributes.

2.4.1 Large-fan-in Boolean gates. When homomorphically evaluating a Boolean function represented as a Boolean circuit, fewer gates in the circuit result in fewer FBS operations involved in the homomorphic computation. Circuit area is an important quality measure in digital circuit synthesis, also known as *logic synthesis*. Thus, the task of homomorphic Boolean circuit synthesis can be viewed as a special case of area-oriented logic synthesis, where each gate has a unitary cost, and the optimization objective is specified as the gate count. Due to this, TFHE compilers in the literature, such as `ROMEO` [15] and `Transpiler` [14], tend to leverage existing logic synthesis tools, such as `Yosys` [25], to convert the Boolean circuit into a network consisting of gates from the gate set supported by the `TFHE` library, a process known as *technology mapping*.

In the `TFHE` library, only the homomorphic evaluation of two-input logic gates and three-input multiplexers is implemented. However, existing research reveals that certain gates with larger fan-in sizes, such as full adders, can be homomorphically evaluated as efficiently as the gates supported by the `TFHE` library [18]. This presents an opportunity to achieve higher-performance homomorphic Boolean circuit designs. By utilizing the expressiveness of large-fan-in gates, more compact circuits can be synthesized.

Despite this potential, there is a lack of exploration regarding the question: "What kind of large-fan-in logic gates can be supported?" Due to the limited study on this problem, existing efforts to exploit large-fan-in gates for compact TFHE circuit designs have taken two directions. The first direction involves considering a limitedly extended gate set, with a significantly limited number of large-fan-in-size gates added to the original `TFHE` gate set. For instance, only full adders are considered in [20]. While this approach allows formulating the TFHE circuit synthesis problem as a technology mapping problem and benefits from the expressiveness of large-fan-in gates, it suffers from the limited number of considered large gates.

The second direction starts with a Boolean circuit consisting of two-input gates and locally *compounds* concatenated two-input gates into large fan-in ones, validating each compounded gate on the fly. However, even though greedily compounding concatenated two-input gates, as proposed in [16], maximizes the use of large-fan-in gates, the lack of a global view means this does not necessarily lead to compact circuit designs, as we will later report in our experiments (Section 5). To address this limitation, [5] proposes keeping several local compounding solutions and deciding which to commit at the last moment to minimize the gate count. Nevertheless, the greedy nature of compounding suggests that technology mapping is more promising for unleashing the expressiveness of large-fan-in gates and achieving compact TFHE circuit designs.

2.4.2 Multi-output Boolean gates. The multi-value FBS technique adds an additional dimension to the TFHE circuit synthesis problem. Specifically, it suggests that the cost of a multi-output gate, which implements m single-output Boolean functions sharing the same n inputs (i.e., an n-input m-output Boolean function $f : \mathbb{B}^n \rightarrow \mathbb{B}^m$), is nearly equivalent to that of an n-input single-output gate, provided that the m functions are of the same type and share the same weight assignment (Section 2.2.2).

While the multi-value FBS technique presents an opportunity to achieve lower execution costs in homomorphic Boolean circuit designs, support from the circuit synthesis side is unfortunately very limited. To our knowledge, existing efforts to exploit this technique are limited to performing a post-synthesis *merging* process to combine qualified single-output gates into multi-output ones [16, 20]. However, there is a lack of exploration on integrating this feature into the circuit synthesis process, *i.e.*, developing multi-value-FBS-aware circuit synthesis algorithms to unlock more merging opportunities and fully utilize this technique.

Moreover, reusing existing hardware circuit synthesis algorithms to address this issue can lead to sub-optimal designs, as the feature where a multi-output gate has a cost close to a single-output gate rarely exists in the context of hardware circuits. The nearest counterparts we have noticed are dual-output *look-up tables* (LUTs) for *field-programmable gate array* (FPGA) designs [24] and multi-output standard cells, such as half and full adders, for standard-cell-based digital circuit designs [4]. Nevertheless, in these scenarios, the considered multi-output gates typically have a limited output count of no more than two.

In contrast, TFHE circuit synthesis benefits from the multi-value FBS technique, which applies to logic gates with arbitrary output counts. Specifically, the larger the output count, the more the execution cost is amortized. For instance, our experimental results show that the output count of multi-output gates in our synthesized circuits can be as high as five. This gap indicates that reusing existing logic synthesis algorithms may overlook significant optimization opportunities in homomorphic Boolean circuit synthesis. This underscores the necessity of developing innovative and customized algorithms for this purpose.

3 Homomorphic Boolean Circuit Synthesis via Technology Mapping

Logic gates that implement certain Boolean functions, even with a fan-in size larger than two, can be utilized to synthesize homomorphic Boolean circuits while adhering to stringent security parameters. Leveraging larger-fan-in logic gates allows for the execution of more complex operations within a single gate, resulting in more compact circuit designs. This approach enhances the efficiency of homomorphic evaluation without compromising overall security.

In this section, we aim to achieve this goal by addressing two major challenges: (1) Identifying the set of larger-fan-in logic gates qualified for TFHE circuit synthesis. (2) Leveraging these larger-fan-in logic gates to achieve optimal TFHE circuit design.

3.1 Homomorphic Gate Set

The truth table of a logic gate describes the output of the gate for each potential input pattern. For an n-input gate, the truth table has 2^n entries, corresponding to the 2^n possible input combinations. Symbolically, we denote each entry as $f_{(b_{n-1}\dots b_0)_{10}} \in \mathbb{B}$, indicating the output of the gate for the input combination $x_0 = b_0, \dots, x_{n-1} = b_{n-1}$, where $(b_{n-1}\dots b_0)_{10}$ represents the decimal value of the bit string $b_{n-1}\dots b_0$, with the leftmost bit being the most significant.

In the context of TFHE, the truth table of a logic gate is encoded as the p coefficient segments of a test vector. Recall that each segment encodes the same truth table value because of the rounding functionality of the test vector. Once the security parameters are set, the plaintext space size p is fixed, determining the maximum allowable truth table size. Therefore, the validity of a logic gate depends on the ability to *compress* its truth table such that its size does not exceed p and can fit into a test vector.

Compression essentially involves using one entry to represent multiple input patterns. In TFHE, this is achieved by leveraging two key observations: (1) By devising the weight w assigned to each input variable, some entries with the same output are projected to a single entry. (2) The negacyclic property of test vectors (Section 2.2.1). We present a comprehensive analysis based on Boolean properties to derive a complete set of gates that fit within the plaintext space $\mathbb{Z}_4$. This step is crucial for effective technology mapping, which will be discussed in the subsequent sub-section.

3.1.1 *Symmetric gates.*

Based on the function of a logic gate, a dedicated weight assignment for the input variables may allow some entries with the same output to share the same weighted summation $\sum_{i=0}^{n-1}(w_i \cdot b_i)$. This allows using one entry of a truth table to express multiple entries in the baseline 2^n-entry truth, realizing a compression. Indeed, in the baseline case, each input variable is assigned a power-of-two weight value by default, such that each input combination has a unique weighted summation value that distinguishes it from others.

When compressing a truth table by finding an optimal weight assignment for each input variable, logic gates that implement symmetric functions benefit the most. The output of a symmetric function is determined by the Hamming weight of the input pattern, *i.e.*, each input variable is equal when determining the output. By assigning equal weight to all input variables, there are $n + 1$ possible numeric values of the weighted summation for an n-input gate, with the minimum achieved when all inputs are zero and the maximum when all inputs are one. The magnitude of this maximum determines the noisiness of the FBS operation, so selecting the unit weights helps manage the noise level effectively.

From this perspective, all three-input symmetric gates are feasible for synthesizing homomorphic Boolean circuits under the plaintext space size configuration $p = 4$.

3.1.2 *Negacyclic gates.*

Due to the negacyclic property of the polynomials in the TFHE scheme, where if

$$\exists k \in [2^{n-1}..2^n), \text{s.t. } \forall i \in [k..2^n)\ f_i = \neg f_{i-k},$$

only the first k entries of the truth table need to be encoded into the N coefficients of a test vector, to ensure correct functionality. Given that the baseline truth table size of a three-input logic gate is 8 (2^3), a gate is valid only if there exists a k that achieves its maximum value of 4 (2^{3-1}). In other words, for such a gate, half of its truth table is the negation of the remaining half. We term Boolean gates with such truth tables as *negacyclic gates* and the functions they implement as *negacyclic functions.*

We have the following observations regarding negacyclic Boolean functions.

Theorem 3.1. *An n-variable Boolean function f is negacyclic if and only if f can be decomposed into*

$$x_{n-1} \oplus f',$$

where '$\oplus$' denotes the Boolean XOR operation, $\alpha \in [0..n-1]$, and f' is an $(n-1)$-variable function independent of x_α.

Proof. ($\Rightarrow$) By applying Shannon decomposition, f can be decomposed into

$$x_{n-1} \cdot f(x_{n-1}, \ldots, x_0)|_{x_{n-1}=1} + (\neg x_{n-1}) \cdot f(x_{n-1}, \ldots, x_0)|_{x_{n-1}=0},$$

where '$\cdot$' and '$+$' indicate Boolean AND and OR operations, respectively. Since the truth table of f is negacyclic and x_n is the most significant variable, we have

$$f(x_{n-1}, \ldots, x_0)|_{x_{n-1}=1} = \neg f(x_{n-1}, \ldots, x_0)|_{x_{n-1}=0}.$$

By plugging this equation into the decomposed expression, we obtain the following representation of f:

$$x_{n-1} \oplus f(x_{n-1}, \ldots, x_0)|_{x_{n-1}=0},$$

i.e., x_{n-1} is successfully disjointed from f.

($\Leftarrow$) It is straightforward to see that, as long as we regard the disjoint variable, x_α, as the most significant variable, *i.e.*, assign x_α the weight of 2^{n-1}, any resulting truth table would be negacyclic. □

As Theorem 3.1 suggests, to encode the truth table of a three-input negacyclic gate within the plaintext space size, the disjoint input must be assigned the largest weight, ensuring that the resulting truth table is negacyclic.

Theorem 3.2. *Negating the inputs and output of a negacyclic function does not affect the negacyclicity of the resulting function.*

Proof. Neither input negations nor output negations change the property that the disjoint variable can be detached from the function. Thus, the resulting functions remain negacyclic. □

Theorem 3.2 is applicable in the technology mapping and inverter reduction stages, which will be introduced later.

Based on our demonstrations, we propose a homomorphic gate set for plaintext space $\mathbb{Z}_4$, consisting of three-input symmetric gates, three-input negacyclic gates, and all two-input gates.

3.2 Area-Oriented Technology Mapping

In the technology mapping stage, the target computation is first represented as an initial multi-level logic network of simple gates, called the *subject graph*. In our implementation, we choose XAG as the logic representation for the subject graph. As introduced in Section 2.3, each node in an XAG implements either a two-input AND operation or a two-input XOR operation, with potential negations recorded as an edge attribute.

We believe XAG is a suitable representation of the subject graph for our problem. This is because, in a TFHE circuit, any two-input Boolean gate is allowed. Thus, XAG can provide a compact representation for the subject graph, as any two-variable Boolean function can be represented using one XAG node.

We then transform the XAG into a network consisting of gates from the proposed homomorphic gate set (*i.e.*, a TFHE circcuit). This process involves two steps: (1) *Matching*: Enumerating different allowed gates that match the local functions of each node in the subject graph. (2) *Selection*: Selecting the optimal subset of matches to cover the subject graph, where the optimality is defined as using the minimal number of matches. We will now detail our implementation for these two steps.

3.2.1 Matching. The process begins with cut enumeration [9], where 3-feasible cuts rooted at each node in the subject graph are enumerated. Each cut highlights a portion of the subject graph and represents a possible implementation of its root node. These cuts are then matched against the proposed homomorphic gate set to find suitable solutions for implementing the root node by applying gates from the gate set to nodes at lower logical levels in the subject graph.

The matching process leverages Boolean matching, which involves comparing the local function of each cut to each gate candidate, eliminating those cuts that cannot be realized using a single gate from the gate set. The objective is to accurately bind the cuts to the available primitives, thereby enabling optimal mapping of the subject graph to the target gate set.

To enhance matching opportunities, we extend the gate set to include three-input symmetric gates with one input negation. This approach reasonably increases the likelihood of successful matches. When such a match is detected and committed, the cut in the subject graph is implemented using one symmetric gate and an inverter in the resulting TFHE circuit. Notably, the homomorphic operation corresponding to an inverter, which is a negation, does not require any FBS operation. If input negations are not considered during the matching step, the resulting mapped TFHE circuit is likely sub-optimal, as some local mapping choices would be missed.

It is important to note that not all input or output negations need to be explicitly considered, as they can be redundant. For symmetric gates, due to the properties of symmetric functions, it is unnecessary to consider output negation or input negations at more than half of the inputs. This is because: (1) Output negation does not affect the (a)symmetry of a Boolean function; and (2) Negating n' inputs of an n-input symmetric gate is equivalent to negating the remaining $(n-n')$ inputs and the output. Therefore, it is unnecessary to consider three-input symmetric gates with more than one input negation, nor is it necessary to explicitly consider output negation, as these gates are already accounted for by the current gate set, which includes symmetric gates with one input negation.

For the same reason, during the matching step, we do not explicitly consider potential input and output negation for three-input negacyclic gates (as pointed out by Theorem 3.2) or two-input gates (as the proposed gate set already includes all two-input gates).

This approach ensures that all potential matching opportunities are efficiently and effectively considered during the matching steps, which is a crucial factor in ensuring the quality of the resulting mapped homomorphic Boolean circuits.

3.2.2 Selection. For the selection step, our implementation is based on the approach described in [19]. This step focuses on choosing the best subset of matches identified during the matching step to cover the entire subject graph efficiently. It requires evaluating the

Algorithm 1: Each iter. of selection in technology mapping.

```
Input: Subject graph $G$ and cost estimation heuristic est_cost.
Output: Subject graph $G$, with one rep. cut selected for each node
1 foreach node $n \in G$ in topological order do
2     $n.cost \leftarrow +\infty$
3     foreach matched cut $c \in$ cuts rooted at node $n$ do
4         $c.cost \leftarrow cost\_est(G, c)$
5         if $c.cost < n.cost$ then
6             $n.cost \leftarrow c.cost$
7             $n.rep \leftarrow c$
8 return $G$
```

cost of each matched cut and selecting the combination of matches that optimizes the overall circuit design.

The selection algorithm, outlined in Algo. 1, adopts a dynamic programming approach to iteratively update the representative cut of each node, aiming to minimize the total gate count. This process involves selecting cuts that minimize the overall circuit area and updating the network to reflect these choices. However, accurately measuring the area contribution of each cut poses challenges. Consequently, the process (Algo. 1) is repeated across multiple iterations, employing different heuristics for cost estimation (*est_cost*) to refine the mapping and achieve the best possible design outcome. Specifically, we utilize the *area flow* [17] and the *exact area* [19], two well-recognized and complementary heuristics that estimate the cost of a cut from global and local perspectives, respectively, ensuring a comprehensive evaluation of the circuit design.

Area flow is designed to provide a global view of the network by estimating the effective area of each node. For each cut of a node that has passed the matching step, its area flow cost is calculated by summing the gate count needed to implement the root node on top of the leaves (fixed to one gate per cut in our case, as ensured during the matching step) and the area flow costs of the leaves, divided by the fan-out count of the root node. This heuristic is particularly useful for understanding the shared logic among different logic cones in the subject graph.

The exact area heuristic, on the other hand, focuses on a local view by calculating the precise area contribution of a node to the entire circuit. This heuristic provides an accurate assessment of the local area's impact on the overall area. It involves traversing towards the fan-in side of the current node to determine the exact number of gates required if the cut is selected, specifically measuring the size of its MFFC. One can refer to the subsequent section, where we propose the *enhanced exact area*, for a more detailed illustration of this heuristic.

The selection algorithm (Algo. 1) is executed multiple times, alternating between the two heuristics to iteratively update the representative cuts of each node (*n.rep*) and ensure both global and local optimizations are achieved. The representative cut of each node after the final execution of the algorithm is the cut selected for implementation.

After the selection step, the mapped TFHE circuit is derived from the selected cuts. This is realized by traversing the subject graph in reverse-topological order, *i.e.*, from the PO side to the PI side, replacing each reached node with the corresponding cut chosen during the selection process. Each node's selected cut is used to instantiate a gate from the homomorphic gate set. The resulting network, representing the TFHE circuit, is thus an optimized implementation of the subject graph, depicted with gates from the proposed gate set, ensuring both efficiency and functionality.

4 Multi-value-FBS-aware Mapping

In this section, we elaborate on our proposals for maximizing the use of the multi-value FBS technology [7]. Our support is devised from two perspectives: (1) We integrate the feature that multi-output gates are of the same cost as single-output gates into the technology mapping process, enabling *multi-value-FBS-aware technology mapping*; and (2) We dedicate a post-mapping inverter reduction stage to further increase the chances of applying the multi-value FBS technology.

4.1 Technology Mapping with Label Monitoring

To enhance the technology mapping process introduced in Section 3.2 and unlock more opportunities for merging single-output gates into multi-output ones, we propose and utilize a concept called *label*.

4.1.1 Labels. Enabling multi-value-FBS-aware technology mapping presents a challenge: during the selection step in the technology mapping process, how can we efficiently identify which cuts, once selected, can be instantiated as multi-output gates? Our analysis indicates that this evaluation requires two pieces of information: the leaves and the local function of a cut. Therefore, during the matching step, these two identities are jointly recorded as the so-called label and associated with each cut for later use in the selection step.

Here are the implementation details of each label. Given that the cuts are 3-feasible, meaning each cut has at most three leaves, each label consists of four data points: (a) Three data points indicate the leaf nodes of a cut, canonicalized in topological order with zero-padding if a cut has fewer than three leaves; and (b) One data point indicates the gate type, either symmetric or negacyclic for cuts with three leaves, or an arbitrary gate type for cuts with two leaves. Note that cuts related to other gate types are directly ruled out in the matching step.

Our design of labels incorporates additional detailed strategies to prevent false detection of opportunities to apply the multi-value FBS technique. For labels of cuts with three leaves that implement negacyclic functions, we highlight the disjoint variable by ensuring the corresponding leaf node is listed first and reordering the remaining leaves in topological order. This is crucial because, as revealed by Theorem 3.1, the disjoint variable must be assigned the largest weight to encode the gate's truth table into a test vector. Consequently, even if two cuts have the same leaf nodes and both implement negacyclic functions, they cannot be instantiated as a two-output gate to apply the multi-value FBS technique if the leaf node serving as the disjoint variable is not the same.

Another strategy addresses handling the label when a match requires an input negation. Recall that we consider three-input symmetric functions with one negated variable in the matching step to maximize matching opportunities, as introduced in Section 3.2.1. For the label of such a matched cut, a mark is placed on the data point indicating the leaf node that requires a negation. This is

Algorithm 2: Representative cut selection guided by the enhanced exact area heuristic

```
   Input: Node $n$, Subject graph $G$, and a label counter $label\_count$.
   Output: Subject graph $G$, with one rep. cut selected for node $n$
1  $n.cost \leftarrow +\infty$
2  foreach matched cut $c \in$ cuts rooted at node $n$ do
3      $c.cost \leftarrow ref\_cut(G, c)$
4      if $c.cost < n.cost$ then
5          $n.cost \leftarrow c.cost$
6          $n.rep \leftarrow c$
7      $deref\_cut(G, c)$
8  $ref\_cut(G, n.rep)$
9  $n.ref \leftarrow 1$
10 $label\_count[n.rep.label] \leftarrow label\_count[n.rep.label] + 1$
11 Function $ref\_cut$(subject graph $G$, cut $c$) :
12     $cost \leftarrow 1$
13     if $label\_count[c.label] \neq 0$ then
14         $cost \leftarrow 0$
15     foreach leaf $l$ of cut $c$ do
16         if $l$ is a PI then continue
17         $l.ref \leftarrow l.ref + 1$
18         if $l.ref = 1$ then
19             $cost \leftarrow cost + ref\_cut(G, l.rep)$
20     return $cost$
21 Function $deref\_cut$(subject graph $G$, cut $c$) :
22     foreach leaf $l$ of cut $c$ do
23         if $l$ is a PI then continue
24         $l.ref \leftarrow l.ref - 1$
25         if $l.ref = 0$ then
26             $deref\_cut(G, l.rep)$
```

because, it is necessary to distinguish such a match from one that does not require an input negation, as an input negation necessitates an opposite sign for the corresponding input's weight.

These strategies ensure that the determination of whether several cuts implementing different root nodes can be instantiated as a multi-output gate is straightforward. This determination can be made by checking if the labels of the cuts are identical — If the cuts share the same label, it indicates that they can be implemented as a multi-output gate, whose FBS can be realized by applying the efficient multi-value FBS technique.

4.1.2 Enhanced Exact Area Heuristic. By exploiting labels, efficient and effective multi-value-FBS-aware technology mapping is achieved. During each iteration of the selection step, when the exact area heuristic is used for cost estimation, a label counter is maintained to dynamically monitor the labels of the representative cuts determined so far. It is important to note that in the selection step, the representative cut of each node is determined in a topological order (line 1 in Algo. 1). In our enhanced exact area heuristic, the label counter is utilized to ensure that the labels of the representative cuts of nodes determined earlier, which are at a lower logical level, play a crucial role in the determination of the representative cuts of nodes determined later. Algo. 2 depicts how the representative cut of a node is determined following the proposed enhanced exact area heuristic.

Given a cut candidate c of node n that has passed the evaluation at the matching step, its cost is obtained by simulating the implementation of node n as indicated by cut c and counting the number of additional gates required for this implementation. This is achieved by recursively incrementing the reference counters (*ref*) of the nodes in the transitive fan-in cone of cut c (line 3 in Algo. 2), handled by the *ref_cut* function (lines 11-20). The process continues to the fan-ins of a node as long as the reference counter of this node is zero before incrementation (lines 17-19), indicating that implementing node n following the way indicated by cut c necessitates the implementation of the current node that is not yet implemented.

It is important to note that the need to implement such a node does not necessarily result in additional area costs, due to the possibility of jointly implementing this node and existing nodes as a multi-output gate. To check this possibility, we refer to the label counter (*label_count*) to see if the label of the representative cut of the currently reached node matches the label of the representative cut of any already implemented node. This is the most significant difference between the proposed enhanced exact area heuristic and the original one introduced in Section 3.2.2. If such a match is detected, the implementation of the current node does not result in an additional gate count (lines 13-14). This strategy guarantees a precise measurement of the MFFC size of the current node.

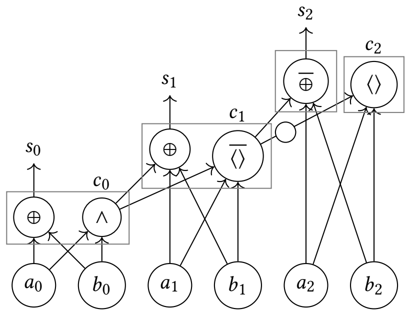
When the currently reached node has a non-zero reference counter or when it is a PI, both of which ensure that this node is already implemented, the exploration towards the fan-ins of this node stops (line 16). At the end of evaluating each cut candidate, the *deref_cut* function is invoked (line 7). This function reverses the incrementation in reference counters done by the *ref_cut* function, enabling the subsequent evaluation of other cuts rooted at node n (lines 21-26).

After evaluating all candidate cuts, based on the cost calculated following the enhanced exact area heuristic, the cut corresponding to the lowest-cost implementation of node n is selected as its representative cut (lines 2-7). The reference counters and the label counter are updated correspondingly, as required for the representative cut selection of subsequent nodes (lines 8-10).

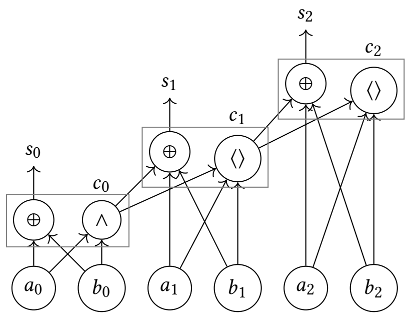
4.2 Inverter Reduction

As discussed in Section 3.2.1, the matching step of the technology mapping process considers potential negations at one input for three-input symmetric gates to ensure comprehensive inclusion of all matching opportunities. While this strategy enhances compact circuit design, it sometimes impedes the application of the multi-value FBS technique. This issue is exemplified as follows.

Example:
Fig. 3a shows the least significant three bits of the 128-bit adder design after technology mapping, where '$\wedge$' indicates Boolean AND operation, '$\langle\rangle$' and '$\overline{\langle\rangle}$' indicate Boolean majority and minority operations, respectively, '$\oplus$' and '$\overline{\oplus}$' indicate Boolean XOR and XNOR operations, respectively. Inverters are denoted using '$\circ$' for visibility, rather than dotted arrows as in Fig. 2. Since node c_1 is a minority gate and inverts the logic of the second carry-out signal, an inverter is necessary before its connection



(a)



(b)

Figure 3: Least significant three bits of the mapped 128-bit adder: (a) Before and (b) After post-mapping inverter reduction. FBS operations for gates highlighted in the same gray box can be jointly conducted via a multi-value FBS operation.

to node c_2 to generate the third carry-out signal. The presence of this inverter prevents the FBS operations of s_2 and c_2 from being realized through a single multi-value FBS operation. Among the 127 pairs of sum and carry-out signals in the 128-bit adder, this issue occurs once for every two pairs, except for the lowest significance pair (s_0 and c_0). Consequently, multi-value FBS is applicable to only 64 pairs, requiring the separate execution of FBS operations for the remaining 63 pairs. Hence, the execution of the TFHE circuit in Fig. 3a involves 191 ($64 + 63 + 63 + 1$) FBS operations. However, this inverter can be eliminated by negating c_1 and s_2's gate functions. Because of the Boolean properties of the gate functions of c_1 and s_2, this elimination does not introduce additional inverters and results in the circuit design shown in Fig. 3b. The TFHE circuit in Fig. 3b is preferable, involving only 128 FBS operations.

Without considering the application of the multi-value FBS technique, both designs in Figs. 3a and 3b are optimal in terms of gate count, as inverters are not counted (since their execution does not require an FBS operation), and both consist of 255 single-output gates. This demonstrates the effectiveness of considering potential input negations during the matching step to achieve compact circuit designs. Further analysis reveals that the design in Fig. 3a, rather than the one in Fig. 3b, results from mapping because the node in the subject graph corresponding to c_2 implements the logic of the negation of the second carry signal. Thus, the circuit design in Fig. 3a is an unavoidable intermediate stage to achieve the design in Fig. 3b. These observations suggest that, rather than eliminating the consideration of input negations during mapping, it is wiser to address inverter reduction as a post-mapping stage.

Below, we introduce our post-mapping inverter reduction approach. After obtaining a mapped logic network, we enumerate the nodes in the network in topological order. As analyzed above, only three-input nodes with a symmetric gate function may result in an inverter at one of their inputs. As noted in Section 3.2.1, output negation does not alter the type of any of the three gate types in the proposed homomorphic gate set. Therefore, when such a node n is encountered, whose negated input node is denoted as l, the inverter can always be absorbed into l's gate function.

However, we shall check the fan-out size of l to decide whether to commit this absorption or not. If node l has only one fan-out, which is node n, the absorption is committed, as the inverter will be reduced without necessitating inverter insertion elsewhere in the network. If node l has more than one fan-out, since the negated gate function at node l will result in inverter insertion when contributing to other fan-outs, the absorption would not be committed until it is confirmed that the newly inserted inverters can be further absorbed into the other fan-outs of l, so that no extra inverter would be inserted into the resulting network. When such a fan-out of l, denoted as node n', has the gate function of either a three-variable negacyclic function or any two-variable function, the confirmation is quickly given, as it is pointed out in Section 3.2.1 that input negation would not alternate the function type of these two. When the gate function of node n' is a three-variable symmetric one, the inverter absorption at node l shall not be committed only if either (1) Node l is also a negated fan-in of node n', or (2) the gate function of Node n' is XOR or XNOR: In case (1), the two inverters on the edge from node l to node n' will cancel each other; In case (2), due to the property of Boolean XOR and XNOR operations, the additional inverter can be absorbed into node n' by negating its gate function. Indeed, the example in Fig. 3 belongs to case (2), where nodes c_2, c_1, and s_2 play the roles of n, l, and n', respectively.

5 Experimental Evaluations

In this section, we introduce the experiments designed and conducted to evaluate the proposed homomorphic Boolean circuit synthesis approach. The experiments are divided into two stages. The first stage focuses on the high-level performance aspects, including the quality measures of the optimized Boolean circuits and the runtime of the circuit synthesis process. The second stage estimates the homomorphic evaluation efficiency of the optimized circuits, providing a precise measurement of the practical gains achieved through our circuit synthesis technique. All experiments are conducted on an Apple M1 Max chip with 32GB memory.

5.1 Profiling Synthesized Circuits

Three homomorphic Boolean circuit synthesis flows are involved in our experimental evaluation:

(1) *SOTA*: Since the state-of-the-art approach we are comparing to, `AutoHoG`, is not open-sourced, we implemented it following the description in [16]. The implemented flow is depicted in Fig. 4a. Initially, area-oriented technology mapping is applied to the subject graph, with all two-input gates constituting the

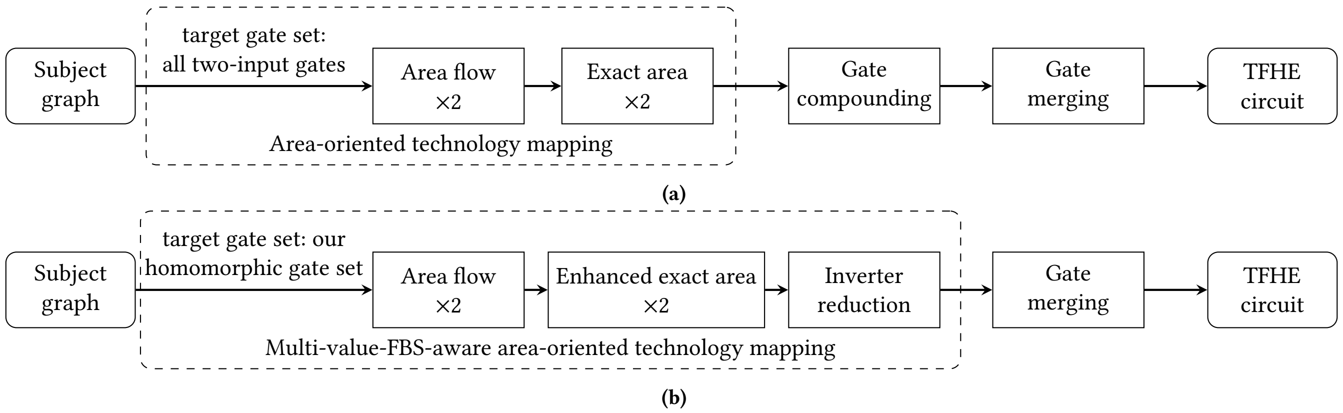


Figure 4: Evaluated homomorphic Boolean circuit synthesis flows: (a) State-of-the-art and (b) Ours.

target gate set. Then, the utilization of the three-input gates in the homomorphic gate set proposed in Section 3.1 is achieved by locally checking if two concatenated two-input gates can be *compounded* into a three-input one, known as the *gate compounding* technique. Finally, in the *gate merging* stage, single-output gates with the same support and the same type of gate functions are *merged* into multi-output ones, to facilitate the application of the multi-value FBS technique.

(2) *TechMap*: To solely evaluate the gain obtained by exploiting larger-fan-in gates via technology mapping, rather than gate compounding, this flow is derived by tuning *SOTA* to integrate our proposal in Section 3. Specifically, the proposed homomorphic gate set, rather than only two-input gates, is adopted as the target gate set for technology mapping. In this way, the gate compounding stage in *SOTA* is not included in this flow.

(3) *Enhanced TechMap*: As depicted in Fig. 4b, this flow integrates all our proposals and differs from *SOTA* in two main aspects: (i) As in *TechMap*, the utilization of three-input gates is achieved by exploiting the proposed homomorphic gate set as the target gate set in the technology mapping stage. (ii) The area-oriented technology mapping is enhanced to be multi-value-FBS-aware by replacing the exact area heuristic with the proposed enhanced version (Section 4.1) and integrating the proposed post-mapping inverter reduction technique (Section 4.2).

The flows are evaluated using the EPFL combinational benchmark suite [1], which consists of ten arithmetic benchmarks and ten random/control benchmarks. As introduced in Section 3, we chose to represent the subject graphs as XAGs for a compact representation. Therefore, as a pre-processing step, the initial benchmarks, provided in *AND-inverter graph* (AIG) representation, are converted to XAGs using the mapper described in [23] before being fed into the flows. All three circuit synthesis flows, as well as the pre-processing, are implemented as part of the C++ logic synthesis library `mockturtle` [22].

The evaluation results are presented in Table 1, which reports the number of gates after merging in each synthesized homomorphic Boolean circuit design ('#Gates'), the merging rate ('Merge'), and the runtime of each circuit synthesis flow ('Time'). The merging rate is defined as the reduction in gate count achieved by merging single-output gates into multi-output ones in the gate merging step. Recall that the gate count of a homomorphic Boolean circuit directly correlates with its execution time, as each Boolean gate corresponds to an FBS operation.

Comparing the designs synthesized by *SOTA* and *TechMap*, since the only difference between these two flows is the method of utilizing large-fan-in gates, the observation that circuits synthesized by *TechMap* consistently have fewer gates, if not the same number, as those generated by *SOTA*, strongly supports our argument that the gate compounding technique in *SOTA* [16] suffers from a lack of a global perspective and results in sub-optimal circuit designs. Although not included in Table 1 due to space constraints, it is noteworthy that circuits synthesized by *SOTA* consistently involve more three-input gates, with an average of 1.92× more than those synthesized by *TechMap*. This further illustrates that while the gate compounding technique maximizes the use of large-fan-in gates by greedily compounding concatenated two-input gates into three-input ones, it does not fully exploit the expressiveness of large-fan-in gates to produce more compact homomorphic Boolean circuits and, ultimately, more efficient homomorphic evaluation. In contrast, our solution of including large-fan-in gates into a dedicated homomorphic gate set for technology mapping yields better designs, as evidenced by the 27.08% average reduction in gate count. Furthermore, the gate compounding technique relies on instantiating a large-fan-in gate to replace concatenated two-input ones whenever an opportunity arises, resulting in inferior runtime performance compared to our technology-mapping-based approach. In our approach, all decisions in the cut selection are made on the subject graphs (Section 3.2.2) and the resulting TFHE circuits are derived in a single pass, leading to an average 33.90% reduction in circuit synthesis time.

Comparing *Enhanced TechMap* to *TechMap*, the additional label monitoring strategy involved in the enhanced exact area heuristic increases the runtime overhead by 1.15×, but it remains lower than *SOTA*, with an average reduction of 26.27%. The introduction of label monitoring contributes to a consistently higher merging rate and, most importantly, more compact circuit designs. *Enhanced TechMap* achieved an additional 4.20% reduction in gate count over *TechMap*, resulting in a 30.15% gate count reduction compared to

Table 1: Evaluating the three homomorphic circuit synthesis flows.

Benchmark	*SOTA*			*TechMap*			*Enhanced TechMap*		
	#Gates	Merge[%]	Time[s]	#Gates	Merge[%]	Time[s]	#Gates	Merge[%]	Time[s]
adder	507	20.16	<0.01	191	25.39	<0.01	128	50.00	<0.01
barrel shifter	2 496	7.45	0.01	2 496	7.45	0.01	2 496	7.45	0.01
divider	28 773	0.10	0.08	13 116	0.31	0.06	13 076	0.82	0.06
hypotenus	121 325	3.72	0.55	83 194	8.82	0.36	78 076	14.56	0.40
log2	20 160	2.09	0.05	14 058	6.34	0.05	13 573	9.62	0.06
max	2 154	5.77	0.01	2 068	8.70	0.01	2 066	11.41	0.01
multiplier	14 530	3.74	0.04	10 442	8.60	0.03	9 957	12.85	0.04
sine	3 578	1.65	0.01	2 502	6.64	0.01	2 398	10.92	0.01
square-root	10 452	1.19	0.03	8 276	17.37	0.03	8 218	17.95	0.03
square	11 968	2.09	0.03	8 661	7.12	0.02	7 547	19.11	0.03
round-robin arbiter	11 434	0.00	0.06	11 605	0.00	0.07	11 605	0.00	0.08
coding-cavlc	554	9.62	<0.01	545	8.86	<0.01	542	11.15	<0.01
ALU control unit	97	11.82	<0.01	94	8.74	<0.01	94	10.48	<0.01
decoder	291	3.96	<0.01	292	2.34	<0.01	292	3.95	<0.01
i2c controller	1 109	2.89	<0.01	1 032	2.46	<0.01	1 029	3.38	<0.01
int to float converter	175	8.85	<0.01	171	8.56	<0.01	170	11.46	<0.01
memory controller	36 446	2.70	0.28	35 510	2.96	0.12	35 016	5.45	0.13
priority encoder	833	0.12	<0.01	818	0.12	<0.01	818	0.24	<0.01
look-ahead XY router	174	1.14	<0.01	134	4.96	<0.01	126	11.27	<0.01
voter	5 170	12.55	0.03	3 306	23.98	0.01	2 936	33.15	0.01
Average	13 611.30	5.08	0.059	9 925.55	7.99	0.039	9 508.15	12.26	0.435
Norm.	**1**	**1**	**1**	**0.729**	**1.572**	**0.661**	**0.699**	**2.413**	**0.737**

SOTA. Interestingly, although the corresponding data is not included in Table 1 due to space constraints, we observed that before the gate merging stage, the average gate count of circuits synthesized by *Enhanced TechMap* is slightly higher (0.36%) than those generated by *TechMap*. This suggests that focusing exclusively on the gate count as the optimization objective during the circuit synthesis process can lead to sub-optimal circuit designs after gate merging. This underscores the importance of developing multi-value-FBS-aware technology mapping techniques to fully utilize the multi-value FBS technique and achieve high-performance homomorphic evaluation.

5.2 Estimating Execution Cost Reduction

The previous experiment provides a coarse-grained evaluation of the speedup in homomorphic evaluation by simply measuring the reduction in gate count achieved by the optimized TFHE circuit designs. It assumes that each homomorphic gate has the same execution cost across different circuit designs, which is not accurate. While we configured the plaintext space as $\mathbb{Z}_4$, the execution cost of each homomorphic gate further depends on the types of Boolean gates used in a TFHE circuit. This, in turn, determines the parameter selection for a sufficiently large noise budget, ultimately affecting the cost of each FBS operation in this circuit.

For example, the 128-bit adder circuit design optimized by *Enhanced TechMap*, as shown in Fig. 3b, consists of one half adder on its least significant bit, which comprises one two-input AND gate and one two-input XOR gate, and 127 one-bit full adders on the remaining bits, each consisting of one three-input majority gate and one three-input XOR gate. Since all these gates are symmetric, assigning a unit weight to every variable results in a plaintext space of $\mathbb{Z}_3$ for the homomorphic evaluation of a half adder and $\mathbb{Z}_4$ for a full adder. More importantly, after being compressed by exploiting symmetry, the 4-entry truth tables of a three-input majority gate and a three-input XOR gate are 1100 and 1010, respectively, with the leftmost bit representing the output when the three inputs are all ones. Notice that both truth tables feature the leftmost bit as the negation of the rightmost bit, indicating the feasibility of exploiting the negacyclic property to further reduce the required plaintext space to $\mathbb{Z}_3$. Therefore, compared to other optimized benchmarks that require a plaintext space of 4, the security parameter for homomorphically evaluating the adder circuit is more relaxed, determining that each FBS operation has a lower computational overhead. This fact is not considered in the previous experiment.

This experiment aims to provide a more reliable measurement of the achieved improvement in homomorphic evaluation efficiency. We use the TFHE compiler `Concrete` [26] to estimate the execution cost per FBS operation in each synthesized circuit. The FBS execution cost mainly depends on the plaintext space size and the Euclidean norm of the linear combination. In the case of a multi-value FBS operation, the Euclidean norm is scaled up with a corresponding factor; Based on the resulting Euclidean norms, `Concrete` selects the parameters to allocate an appropriate noise budget, providing a reliable execution cost estimation of each FBS operation. The estimated execution cost of each synthesized circuit[2]

[2]Our execution cost estimator is available on: https://github.com/ssmiler/tfhe_lbf_eval

Table 2: Estimated execution cost of the synthesized homomorphic Boolean circuits.

Benchmark	*SOTA*	*TechMap*	*Enhanced TechMap*
	Exec. cost	Exec. cost	Exec. cost
adder	20 280	7 258	4 864
barrel shifter	99 840	99 840	99 840
divider	1 150 920	524 640	523 040
hypotenus	4 853 000	3 327 760	3 123 040
log2	806 400	562 320	542 920
max	86 160	82 720	82 640
multiplier	581 200	417 680	398 280
sine	157 432	110 088	105 512
square-root	418 080	331 040	328 720
square	478 720	346 440	301 880
round-robin arbiter	457 360	464 200	464 200
coding-cavlc	22 160	21 800	21 680
ALU control unit	3 880	3 760	3 760
decoder	11 640	11 680	11 680
i2c controller	44 360	41 280	41 160
int to float converter	7 000	6 840	6 800
memory controller	1 603 624	1 562 440	1 540 704
priority encoder	33 320	32 720	32 720
look-ahead XY router	6 960	5 360	5 040
voter	227 480	145 464	117 440
Average	553 490.80	405 266.50	387 796.00
Norm.	**1**	**0.732**	**0.701**

is obtained by multiplying the execution cost per FBS operation by the gate count of the circuit.

As reported in Table 2 as 'Exec. cost,' an average reduction of 29.94% in execution cost is achieved by circuits synthesized by *Enhanced TechMap* compared to *SOTA*, with the maximum reduction achieved for the *adder* benchmark (76.02%). This demonstrates the effectiveness of the proposed homomorphic circuit synthesis flow in improving the efficiency of homomorphic evaluation.

6 Discussion

In this section, we discuss key observations and insights from our study, and explore potential future directions.

6.1 Strategic Use of Incomplete Gate Sets

In this paper, we extend the original gate set of the TFHE library by incorporating symmetric gates and nega-cyclic gates. While the proposed homomorphic gate set is complete with the plaintext space $\mathbb{Z}_4$, it is no longer complete when a larger plaintext space size is configured. As pointed out in [16], adopting a larger plaintext space size dramatically increases the number of valid gate types, making the derivation of a complete gate set impractical. This is why AutoHoG is designed to exploit large-fan-in gates by locally compounding small-fan-in ones.

However, it is crucial to emphasize that the ultimate goal of exploiting large-fan-in gates is to leverage their expressiveness to obtain more compact homomorphic Boolean circuits. The objective is not to maximize the number of large-fan-in gates in the resulting circuits but to use them wisely to minimize the gate counts of the resulting circuits. Due to the greedy nature of gate compounding, the circuit designs delivered by AutoHoG tend to suffer from this misunderstanding. As evidenced by our observation in Section 5.1, circuits synthesized by AutoHoG utilized more three-input gates, while circuits generated by our approach are more compact.

Therefore, we argue that being able to exploit large-fan-in gates smartly, even if not all of them, is more important to the homomorphic Boolean circuit synthesis problem than being capable of exploiting all valid types of large-fan-in gates. Our homomorphic gate set serves as an effective solution, as it includes a sufficient number of large-fan-in gates by considering the Boolean properties of symmetry and negacyclicity. This facilitates technology mapping to realize strategic exploitation of their expressiveness, leading to more compact circuits and more efficient homomorphic evaluation.

6.2 Exploring a Unique Logic Synthesis Problem

As introduced in Section 2.4, while existing TFHE compiler designs tend to adopt hardware circuit synthesis tools for TFHE circuit synthesis, the unique feature that multi-output gates and single-output gates can have similar costs distinguishes this problem as a unique and self-contained logic synthesis challenge. This distinction is technically interesting and warrants further study.

In our multi-value-FBS-aware area-oriented technology mapping algorithm, only the exact area heuristic is enhanced to support the multi-value FBS technique, while the area flow heuristic remains unchanged (Fig. 4b). As introduced in Section 3.2.2, the role of the area flow heuristic is to provide a global view when deciding which matches to select to cover the subject graph, at an early stage of selection. We attempted to modify the area flow heuristic as well, but this did not yield better circuit designs. We believe this is because, at an early stage of the selection step, it is challenging to predict which matches will ultimately be selected, making it difficult to effectively increase the gate merging opportunities. This explains why our enhanced exact area heuristic is effective; it is invoked after the area flow heuristic to locally refine match selection. Since most decisions in cut selection are made at this stage, it becomes straightforward to adjust some selections to increase gate merging opportunities. This suggests that a well-designed enhancement of the area flow heuristic could unlock further optimization, which we aim to explore in future work.

7 Conclusion

In this paper, we present an approach to synthesizing homomorphic Boolean circuits by incorporating larger fan-in Boolean gates and exploiting the multi-value functional bootstrapping technique. Our proposed synthesis flow outperforms the state-of-the-art in both synthesis speed and the quality of the synthesized circuits, achieving significant improvements in homomorphic evaluation efficiency. This work addresses the limitations of existing methods and sets a new standard for future research in FHE.

Acknowledgment

This project is supported in part by Synopsys Inc. We are grateful to the anonymous reviewers for their insightful comments.

References

[1] Luca Amarú, Pierre-Emmanuel Gaillardon, and Giovanni De Micheli. 2015. The EPFL Combinational Benchmark Suite. In *IWLS*.

[2] Guillaume Bonnoron, Léo Ducas, and Max Fillinger. 2018. Large FHE Gates from Tensored Homomorphic Accumulator. In *AFRICACRYPT*. 217–251.

[3] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. 2012. (Leveled) Fully Homomorphic Encryption without Bootstrapping. In *Innovations in Theoretical Computer Science Conference*. 309–325.

[4] Alessandro Tempia Calvino and Giovanni De Micheli. 2023. Technology Mapping Using Multi-Output Library Cells. In *ICCAD*. 1–9.

[5] Sergiu Carpov. 2024. A Fast Heuristic for Mapping Boolean Circuits to Functional Bootstrapping. Cryptology ePrint Archive, Paper 2024/1204.

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

[7] Sergiu Carpov, Malika Izabachène, and Victor Mollimard. 2019. New Techniques for Multi-value Input Homomorphic Evaluation and Applications. In *CT-RSA*. 106–126.

[8] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2020. TFHE: Fast Fully Homomorphic Encryption Over the Torus. *Journal of Cryptology* 33 (2020), 34–91.

[9] Jason Cong, Chang Wu, and Yuzheng Ding. 1999. Cut Ranking and Pruning: Enabling a General and Efficient FPGA Mapping Solution. In *FPGA*. 29–35.

[10] Léo Ducas and Daniele Micciancio. 2015. FHEW: Bootstrapping Homomorphic Encryption in Less Than a Second. In *EUROCRYPT*. 617–640.

[11] Junfeng Fan and Frederik Vercauteren. 2012. Somewhat Practical Fully Homomorphic Encryption. Cryptology ePrint Archive, Paper 2012/144.

[12] Craig Gentry. 2009. Fully Homomorphic Encryption using Ideal Lattices. In *Annual ACM Symposium on Theory of Computing*. 169–178.

[13] Craig Gentry, Amit Sahai, and Brent Waters. 2013. Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In *CRYPTO*. 75–92.

[14] Shruthi Gorantala, Rob Springer, Sean Purser-Haskell, William Lam, Royce Wilson, Asra Ali, Eric P. Astor, Itai Zukerman, Sam Ruth, Christoph Dibak, Phillipp Schoppmann, Sasha Kulankhina, Alain Forget, David Marn, Cameron Tew, Rafael Misoczki, Bernat Guillen, Xinyu Ye, Dennis Kraft, Damien Desfontaines, Aishe Krishnamurthy, Miguel Guevara, Irippuge Milinda Perera, Yurii Sushko, and Bryant Gipson. 2021. *A General Purpose Transpiler for Fully Homomorphic Encryption*. Technical Report. Google LLC.

[15] Charles Gouert and Nektarios Georgios Tsoutsos. 2020. Romeo: conversion and evaluation of HDL designs in the encrypted domain. In *DAC*.

[16] Zhenyu Guan, Ran Mao, Qianyun Zhang, Zhou Zhang, Zian Zhao, and Song Bian. 2024. AutoHoG: Automating Homomorphic Gate Design for Large-Scale Logic Circuit Evaluation. *IEEE TCAD* 43, 7 (2024), 1971–1983.

[17] Valavan Manohararajah, Stephen D. Brown, and Zvonko G. Vranesic. 2006. Heuristics for Area Minimization in LUT-Based FPGA Technology Mapping. *IEEE TCAD* 25, 11 (2006), 2331–2340.

[18] Kotaro Matsuoka, Yusuke Hoshizuki, Takashi Sato, and Song Bian. 2021. Towards Better Standard Cell Library: Optimizing Compound Logic Gates for TFHE. In *WAHC*.

[19] Alan Mishchenko, Sungmin Cho, Satrajit Chatterjee, and Robert Brayton. 2007. Combinational and Sequential Mapping with Priority Cuts. In *ICCAD*. 354–361.

[20] Johannes Mono, Kamil Kluczniak, and Tim Güneysu. 2023. Improved Circuit Synthesis with Amortized Bootstrapping for FHEW-like Schemes. Cryptology ePrint Archive, Paper 2023/1223.

[21] Ronald L. Rivest, Len Adleman, and Michael L. Dertouzos. 1978. On Data Banks and Privacy Homomorphisms. *Foundations of Secure Computation* 4, 11 (1978), 169–180.

[22] Mathias Soeken, Heinz Riener, Winston Haaswijk, Eleonora Testa, Bruno Schmitt, Giulia Meuli, Fereshte Mozafari, Siang-Yun Lee, Alessandro Tempia Calvino, Dewmini Sudara Marakkalage, and Giovanni De Micheli. 2022. The EPFL Logic Synthesis Libraries. arXiv:1805.05121

[23] Alessandro Tempia Calvino, Heinz Riener, Shubham Rai, Akash Kumar, and Giovanni De Micheli. 2022. A Versatile Mapping Approach for Technology Mapping and Graph Optimization. In *ASPDAC*. 410–416.

[24] Feng Wang, Liren Zhu, Jiaxi Zhang, Lei Li, Yang Zhang, and Guojie Luo. 2020. Dual-Output LUT Merging during FPGA Technology Mapping. In *ICCAD*. 1–9.

[25] Claire Wolf. [n. d.]. Yosys Open SYnthesis Suite. https://yosyshq.net/yosys/.

[26] Zama. 2022. Concrete: TFHE Compiler that Converts Python Programs into FHE Equivalent. https://github.com/zama-ai/concrete.