Making the Best Switch: Encoding Strategy Management for Efficient TFHE Circuit Evaluation

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

Gabrielle De Micheli
University of California, San Diego
La Jolla, California, USA
gdemicheli@ucsd.edu

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

***Abstract*—This work addresses the synthesis of efficient *torus fully homomorphic encryption* (TFHE) circuits for private Boolean function evaluation through *encoding strategy management*. Modern TFHE implementations support multiple plaintext encoding spaces, each offering distinct trade-offs between computational cost and the expressiveness enabled by larger plaintext domains. Smartly switching between encoding strategies to maximize evaluation efficiency remains challenging due to the lack of algorithmic support for determining when and where such transitions should occur. To address this, we propose a synthesis framework that enables *encoding-switch-aware TFHE circuit generation*. Our approach leverages the structural properties of the *exclusive-or sum of products* (ESOP) representation to partition Boolean functions into encoding-aligned regions, enabling cost-effective evaluation while minimizing switch overhead. Experimental results demonstrate that our encoding-aware synthesis technique significantly accelerates homomorphic Boolean function evaluation – achieving up to 53.46% and 23.34% average evaluation time reduction on general-purpose Boolean benchmarks – compared to advanced synthesis baselines lacking explicit encoding-switch management. This work lays the groundwork for systematic encoding strategy management in TFHE circuits and highlights the role of logic-level design automation in advancing efficient homomorphic evaluation.**

***Index Terms*—Fully Homomorphic Encryption, Programmable Bootstrapping, Boolean Function Evaluation, Logic Synthesis**

I. Introduction

Fully homomorphic encryption (FHE) enables computation on encrypted data without decryption, offering strong privacy for applications such as secure analytics, outsourced computation, and encrypted machine learning. Despite its compelling security guarantees, FHE remains costly compared to plaintext processing, prompting extensive research into improving its practicality through algorithmic and compiler-level optimizations.

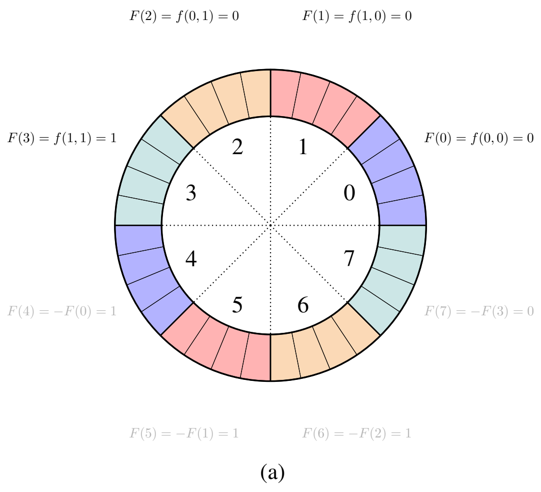
Modern FHE schemes fall into two broad categories: *leveled schemes*, such as BFV [1] and BGV [2], and *fast-bootstrapping schemes*, such as FHEW [3] and TFHE [4]. Leveled schemes support SIMD-style data packing and benefit from optimizations that reduce circuit depth and manage noise growth. In contrast, fast-bootstrapping schemes lift depth constraints by enabling efficient noise-refreshing through frequent *programmable bootstrapping* (PBS), which evaluates *look-up tables* (LUTs) representing small functions. We note that orthogonal to these are schemes like CKKS [5], which support approximate arithmetic over real numbers and are designed for a different class of applications.

In this work, we focus on TFHE as a representative fast-bootstrapping scheme and target a core design question in TFHE circuit synthesis: *how to manage encoding strategies* – i.e., the choice of plaintext space – during homomorphic Boolean function evaluation. When synthesizing TFHE circuits for a given Boolean function, designers can choose among different plaintext encoding spaces (e.g., $\mathbb{B} = \{0, 1\}$, $\mathbb{Z}_4 = \{0, 1, 2, 3\}$, etc.), each offering distinct trade-offs between bootstrapping cost and the ability to evaluate more complex logic gates. Larger encoding spaces support the evaluation of higher-fanin gates in a single PBS, reducing gate count, but at the cost of more expensive bootstrapping operations. Conventional approaches typically select a single encoding space and retain it throughout the entire circuit evaluation. More recently, researchers explored switching between encoding strategies within a single circuit – using $\mathbb{B}$ to exploit *free-XOR* (i.e., homomorphic XOR evaluation without invoking a PBS), and larger spaces to reduce the number of gates required for non-linear logic [6]. While this approach shows promise, its practical adoption remains limited by the lack of algorithmic support for identifying when and where such switches should occur. Moreover, each encoding switch must itself be realized via an additional PBS operation, introducing a new trade-off between switch overhead and evaluation efficiency. Without a cost model or synthesis strategy to guide the placement of encoding-space transitions and restructure the boundary between linear and non-linear components, the potential benefits of this approach remain difficult to realize in general-purpose circuits.

In this paper, we propose a synthesis framework for *encoding-switch-aware TFHE circuit generation*, aimed at improving the efficiency of general Boolean function evaluation. The key contributions of this work are as follows:

- Cost model for TFHE circuit evaluation: We formalize a quantitative cost model that captures both compute and switch PBS operations based on gate type and encoding transitions. This model fills a critical gap in the literature by enabling systematic trade-off analysis for encoding strategy management.
- Structural abstraction via ESOP representation: We leverage the *exclusive-or sum of products* (ESOP) form as a structural abstraction to expose clean partitions between linear (XOR) and non-linear (AND) logic, aligning naturally with our target dual-domain homomorphic evaluation model.
- Cost-guided synthesis algorithm: Building on the ESOP abstraction and cost model, we design a synthesis algorithm that decomposes circuits into encoding-aligned regions and determines optimal switching points to reduce evaluation cost.
- Comprehensive empirical validation: Experimental results on 25 general-purpose Boolean benchmarks demonstrate that our synthesis approach outperforms both fixed-encoding and advanced multiplicative-complexity-reduction baselines without explicit encoding-switch management. Our method achieves up to 53.46% and an average of 23.34% reduction in homomorphic evaluation time.

The remainder of this paper is organized as follows. Section II introduces the necessary background, including an overview of TFHE circuit synthesis, Boolean circuit notations, and a survey of advanced techniques for efficient homomorphic evaluation of Boolean functions. Section III analyzes when and why encoding transitions are needed, focusing on two logic representations: XAG and ESOP. Section IV presents our ESOP-guided synthesis framework, detailing the high-level flow, cost modeling for ESOP evaluation, and the cut



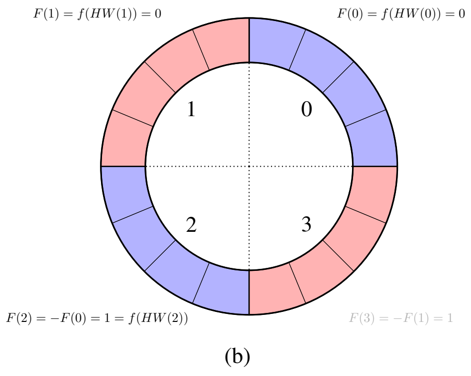


Fig. 1: Encoding the LUT of a two-input AND gate onto a polynomial ring. (a) Naïve encoding using plaintext space $\mathbb{Z}_8$ and polynomial degree $N = 16$. (b) Optimized encoding using plaintext space $\mathbb{Z}_4$ and $N = 8$. Dotted lines mark plaintext space boundaries. Colored segments denote distinct LUT entries; each pair of freely encoded and negacyclically determined (negated) entries shares a color. Entries never indexed by design are faded to reflect their irrelevance during evaluation.

selection methodology. Section V reports experimental results across a wide range of benchmarks. Finally, Section VI concludes the paper and outlines directions for future work.

II. Background and Motivation

A. TFHE Circuit Synthesis: Fundamental and Prior Work

a) TFHE circuits and programmable bootstrapping: We define a *TFHE circuit* as a Boolean circuit composed of logic gates, which serves as a blueprint for evaluating the corresponding Boolean function over encrypted data using the TFHE scheme.

In a TFHE circuit, each logic gate is privately evaluated via a *programmable bootstrapping* (PBS) operation. Due to space constraints, the substantial prerequisite knowledge for a formal exposition, and the fact that this work introduces no new cryptographic primitives, we offer an informal yet intuitive overview of this mechanism. For clarity, we describe operations as if they were performed on plaintext values, though all operands in homomorphic evaluation are encrypted *torus learning with error* (TLWE) ciphertexts. We refer readers to the original TFHE paper [4] and subsequent optimizations, such as [7], [8], for a rigorous treatment.

Conceptually, a PBS operation realizes the homomorphic equivalent of evaluating a Boolean gate by indexing into its truth table. This process can be decomposed into two logical stages: (1) computing the index corresponding to the current input pattern; and (2) using that index to query the truth table and retrieve the expected output.

In the homomorphic setting, this mechanism operates over encrypted inputs (`TLWE` ciphertexts). Consider an n-input Boolean gate implementing a function $f : \mathbb{B}^n \mapsto \mathbb{B}$. Its truth table comprises 2^n entries, each representing the gate's output for a specific input pattern. To evaluate this gate homomorphically, we compute an index via a *projection function* $\phi : \mathbb{B}^n \mapsto \mathbb{Z}_p$ that maps each input pattern to an index in the plaintext space of size p. Accordingly, the encoded *look-up table* (LUT) must implement a function $F : \mathbb{Z}_p \mapsto \mathbb{B}$, such that the correctness condition

$$f(b_1, \ldots, b_n) = F(\phi(b_1, \ldots, b_n))$$

holds for all input patterns. This relation ensures that querying the encrypted LUT via the projected index yields the correct encrypted output.

Once the projection index is computed, the next step is to query the corresponding output from the gate's truth table. To facilitate this, the LUT is encoded onto a degree-N polynomial ring $\mathbb{T}[X]/(X^N + 1)$, where $\mathbb{T}$ denotes the torus. The choice of the irreducible polynomial $X^N + 1$ implies that the polynomial ring is *negacyclic*, i.e., it satisfies the constraint $X^N = -1$. As a result, the ring provides $2N$ coefficient slots that can be used to hold LUT values, but only N of them can be freely assigned – the remaining N are determined by negacyclic symmetry. More formally, for any index $\varphi > \frac{p}{2} - 1$, the LUT must satisfy $F(\varphi) = -F(\varphi - \frac{p}{2})$ to ensure correctness under negacyclic rotation. Each LUT entry $F(\varphi)$ is typically replicated across $\frac{2N}{p}$ consecutive coefficients of the ring to ensure consistent encoding and uniform noise distribution. This allows the use of programmable bootstrapping to query the correct entry via the so-called *blind rotation* process over encrypted inputs.

When no compression is applied, the LUT size is 2^n for an n-input logic gate, and the default projection function is the power-of-two weighted sum:

$$\phi(b_1, \ldots, b_n) = \sum_{i=1}^{n} 2^{i-1} \cdot b_i,$$

which guarantees a unique index for each input pattern.

Fig. 1a illustrates this naïve LUT encoding for a two-input AND gate, using $\phi(b_1, b_2) = b_1 + 2b_2$ to map input patterns to indices in $\mathbb{Z}_8$. The polynomial ring is instantiated with $N = 16$, so each LUT entry is encoded using four coefficients, evenly spaced along the ring. Because the highest index produced by ϕ is 3, all accessed indices fall within the first half of the ring, where entries can be freely assigned. The remaining indices are never queried during evaluation and thus act as *don't-care* values, meaning their contents have no impact on the correctness of the result.

b) Advances in TFHE circuits synthesis: Two major techniques have emerged to achieve compact TFHE circuit designs. The first line of work observes that larger multi-input gates can be accommodated through proper *LUT compression* techniques and parameter tuning, i.e., adopting a larger plaintext space. As discussed earlier, a logic gate with n inputs typically requires a LUT of size 2^n to distinguish all input patterns. However, when multiple patterns yield the same output, the LUT can be compressed by merging entries without affecting correctness.

Among the body of work exploring this direction [9]–[11], we highlight the contribution of [11], which systematically analyzes the

Boolean properties – specifically, symmetry and negacyclicity – that enable such compression. For a fixed plaintext space size p, this method statically enumerates a gate library of all functions whose LUTs can be compressed accordingly, along with their corresponding projection functions ϕ. This static formulation enables large-gate exploitation to be expressed as a standard technology mapping problem, avoiding the runtime overhead and local suboptimality of greedy gate-merging approaches like [9] and [10]. The authors demonstrate the utility of this method under plaintext space $\mathbb{Z}_8$ [1].

Figure 1b illustrates this compression on a two-input AND gate. AND gates are symmetric for any fan-in – their output depends solely on the Hamming weight of the input. This allows for a uniform-weight projection function,

$$\phi(b_1, \ldots, b_n) = \sum_{i=1}^{n} b_i,$$

reducing the LUT size to $n+1$ entries, as all input patterns with the same Hamming weight w (denoted $HW(w)$) map to the same entry $F(w)$. Furthermore, since $F(n) = -F(0)$ for any AND gate, the negacyclic symmetry enables further compression, allowing the LUT to fit within a plaintext space of size $p = 2n$. In the example of $n = 2$, shown in Fig.1b, this approach achieves the same error resilience as the naïve encoding (Fig.1a), where each LUT entry is backed by four coefficients, while reducing the polynomial degree from $N = 16$ to $N = 8$. This demonstrates why the PBS operation under this optimized encoding is substantially less expensive than in the uncompressed case. In other words, under the same plaintext space size p, this approach enables the use of AND gates with fan-in up to four for TFHE circuit synthesis – a notable improvement over the naïve encoding, which limits designs to two-input ANDs – thereby allowing for potentially more compact circuit implementations.

The second technique we highlight is *multi-value PBS* [12], a notable advancement for improving the efficiency of PBS. This method enables the simultaneous evaluation of multiple gates over the same encrypted input within a single PBS operation. Instead of performing separate PBS calls per gate, it shares the LUT query step by factoring the polynomial rings into a common component and function-specific components. The shared query is applied once, and individual outputs are recovered through lightweight polynomial multiplications. For full technical details, we refer readers to the original paper. From a synthesis standpoint, this technique facilitates the use of *multi-output gates* to reduce the total PBS count. Reference [11] incorporates this optimization during technology mapping by dynamically favoring mapping choices that enable shared-input multi-output gate structures, resulting in more compact TFHE circuits.

B. Multi-Encoding-Space Homomorphic Evaluation

All previously discussed synthesis techniques operate under the implicit assumption that each logic gate must be evaluated via a PBS operation. A promising alternative direction aims to reduce PBS usage by leveraging *leveled operations* – i.e., performing homomorphic computations using efficient TLWE ciphertext operations such as ciphertext addition and scalar multiplication, thereby avoiding expensive PBS invocations.

An early exploration of this idea appears in [13], which observes that, under the plaintext space $\mathbb{B}$, homomorphic XOR can be realized via ciphertext addition, and homomorphic AND via ciphertext multiplication. While the XOR realization is ideal – offering low latency and linearly increasing noise, effectively enabling "free-XOR" – the AND realization via ciphertext multiplication is both computationally expensive and introduces a quadratic noise growth. As a result, a PBS must still follow each AND to refresh the noise, raising concerns about the practicality of this strategy.

[1]The authors refer to the space as $\mathbb{Z}_4$ to emphasize the "free encoding" region unaffected by ring negacyclicity. However, since the projected indices span $\mathbb{Z}_8$, we refer to it as $\mathbb{Z}_8$ (implicitly negacyclic) for consistency with the rest of the discussion.

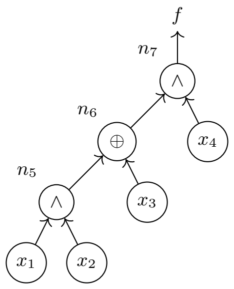


Fig. 2: An XAG implements function $f = ((x_1 \wedge x_2) \oplus x_3) \wedge x_4$.

To retain the efficiency of free-XOR while avoiding costly multiplication-based AND, a natural solution is to evaluate non-linear gates directly using PBS. However, encoding the LUT of a non-linear gate into the limited plaintext space $\mathbb{B}$ – a requirement for free-XOR compatibility – is currently infeasible. This motivates a hybrid strategy: homomorphically evaluate linear gates using leveled operations in $\mathbb{B}$, and evaluate non-linear gates via PBS in a larger plaintext space, noted as $\mathbb{Z}_p$.

Such multi-encoding-space evaluation has been shown to be technically viable. Specifically, reference [6] demonstrates that a PBS operation can simultaneously refresh a ciphertext and switch its plaintext space, by encoding the target ciphertext directly in the desired space within the queried polynomial coefficients. This flexibility introduces new challenges for TFHE circuit synthesis: *When should an encoding switch occur?* and *How can circuit structure be reshaped to minimize unnecessary switches?* We refer to the systematic handling of these questions as *encoding strategy management*. This aspect has been largely overlooked in prior work, despite its critical importance: unnecessary encoding transitions introduce additional PBS operations that can easily negate the performance benefits gained from free-XOR. This paper is the first to formulate and address TFHE circuit synthesis under multi-encoding-space evaluation as a dedicated optimization problem, which we term *encoding-switch-aware TFHE circuit synthesis*.

C. Boolean Circuit Notations

We model a Boolean circuit as a *directed acyclic graph* (DAG) $G = (V, E)$, where each node $v \in V$ represents either a logic gate or a *primary input* (PI), and each directed edge $(u, v) \in E$ denotes a signal wire from node u to node v. For any node $v \in V$, we define its *transitive fan-in* as the set of all nodes from which there exists a directed path to v. The set of primary inputs is denoted by $I \subseteq V$. A subset of nodes – potentially annotated with output negation – forms the set of primary outputs (POs), denoted by $O \subseteq V$. The value of each PO node is treated as an observable output of the circuit. The entire circuit implements a Boolean function $f : \mathbb{B}^{|I|} \mapsto \mathbb{B}^{|O|}$.

a) XOR-AND-inverter graphs: In this work, we focus on *XOR-AND-inverter graphs* (XAGs), a restricted class of Boolean circuits in which each non-PI node has exactly two fan-in edges and implements either a two-input XOR or AND operation. Logic negations are optionally represented as edge attributes on the incoming wires. Fig. 2 illustrates an example XAG implementing a four-variable, single-output Boolean function $f = ((x_1 \wedge x_2) \oplus x_3) \wedge x_4$. It consists of nodes $V = \{x_1, x_2, x_3, x_4, v_5, v_6, v_7\}$, where the first four elements form the PI set I, and the sole PO is $O = \{v_7\}$, with no output negation applied. Each internal gate is annotated with either '$\wedge$' for AND or '$\oplus$' for XOR. XAGs are of particular interest in this work because they cleanly separate non-linear logic (AND gates) from linear logic (XOR gates), a structural property that aligns naturally with the computational asymmetry leveraged in multi-encoding TFHE evaluation – where XOR gates can be evaluated freely in $\mathbb{B}$ and PBS is reserved for non-linear components. This makes XAGs an ideal logic representation for our encoding-aware synthesis framework.

b) Cuts: A *cut* C in a Boolean circuit is a pair (r, L) consisting of a *root node* $r \in V$ and a set of *leaf nodes* $L \subseteq V$, such that every directed path from a PI to r passes through at least one node in L. The subgraph induced between L and r defines a *logic cone*, and the Boolean function computed at r expressed in terms of its leaves is referred to as the *cut function*, denoted $f_C : \mathbb{B}^{|L|} \mapsto \mathbb{B}$. For example, in Fig. 2, consider the cut $C = (n_7, \{n_5, x_3, x_4\})$. This cut isolates a logic cone rooted at n_7 with leaf set $L = \{n_5, x_3, x_4\}$. The corresponding cut function is $f_C = (n_5 \oplus x_3) \wedge x_4$, which is a three-variable Boolean function. A cut is said to be *k-feasible* if it contains at most k leaf nodes. Cuts are widely used to localize subfunctions within circuits and serve as fundamental units in logic synthesis and optimization flows.

III. When to Switch: Analyzing Encoding Strategy Transitions

This section aims to fill the gap in the systematic understanding of encoding strategy management by analyzing when encoding switches are beneficial or necessary. We focus on two circuit representations of interest: XAGs and *exclusive-or sum of products* (ESOP). By examining how structural and functional properties interact with encoding choices, we derive guiding principles for identifying cost-effective encoding transitions – insights that will inform the synthesis algorithms developed in the next section.

In the context of multi-encoding-space homomorphic evaluation, each PBS operation serves one of two distinct roles: it either evaluates a logic gate (referred to as a *compute PBS*) or performs a plaintext encoding space transition. With a slight abuse of terminology, we refer to the latter as a *switch PBS* – a PBS operation whose sole purpose is to convert a `TLWE` ciphertext from one encoding space to another, without performing any logic computation. Although compute PBSs may incidentally trigger an encoding switch, switch PBSs exist solely to enable subsequent operations in a different space. Since both types of PBS operations incur similar computational costs, minimizing the number of switch PBSs is critical for improving the efficiency of encoding-switch-aware TFHE circuit designs.

A. Encoding Strategy Behavior in XAG-Based Evaluation

In this subsection, we analyze how encoding strategy switches arise in circuits represented as XAGs. Recall that XAGs consist of two-input XOR and AND gates, with logic negations encoded as edge attributes. Since XOR operations can be efficiently evaluated in $\mathbb{B}$ using the free-XOR technique, while AND gates require PBS operations, transitions between encoding spaces naturally occur at the boundaries between these gate types.

1) AND Gates with Mixed Fanout: When an AND gate contributes to XOR gates, its output must be in $\mathbb{B}$ to enable efficient downstream evaluation via the free-XOR technique. As before, the need for encoding transitions depends on whether the AND gate also feeds into other AND gates.

a) XOR-Only Fanout: If the AND gate feeds only XOR gates, it serves as the root of its corresponding AND tree. In this case, a compute PBS is applied at the AND gate, producing a `TLWE` ciphertext in $\mathbb{B}$ that can be consumed by XOR gates directly, without requiring any additional space switching.

b) Mixed Fanout: When an AND gate feeds both XOR and AND gates, encoding strategy management becomes more nuanced. Although a compute PBS is still required to produce a ciphertext in $\mathbb{B}$ for XOR evaluation, we have flexibility in how the surrounding AND tree is structured. Two strategies arise: *splitting the AND tree* or *preserving the full AND tree*. These approaches are illustrated in Fig. 3: In Fig. 3a, a representative XAG subnetwork is presented with input and output signals x_i and y_i, respectively; Fig. 3b and 3c show the corresponding homomorphic evaluation flows. For clarity, $c(x_i)$ and $c(y_i)$ denote the `TLWE` ciphertexts of x_i and y_i. Each PBS operation is labeled as either a compute PBS ('PBS(c)') or a switch PBS ('PBS(s)'), and the encoding space of each ciphertext is explicitly indicated on the edges.

In the *split* strategy (Fig. 3b), the lower, mixed-fanout AND gate is treated as the root of a smaller subtree. The lower and upper subtrees are each evaluated using a separate compute PBS. However, since the output of the lower subtree is needed in both $\mathbb{B}$ (for XOR evaluation) and $\mathbb{Z}_p$ (to serve as an input to the upper AND tree), a switch PBS is required to convert between encoding spaces. This results in a total of three PBS operations: two compute PBSs and one switch PBS.

In the *preserve* strategy (Fig. 3c), we treat the entire structure as a single AND tree rooted at the highest AND gate. This approach avoids the need for a switch PBS by maintaining consistent encoding throughout the tree, requiring only two compute PBS operations in total. Although the encoding space must accommodate larger AND trees, the required modulus grows linearly with arity. In practice, the slight increase in compute PBS cost is typically outweighed by the savings from eliminating the switch PBS. Given this trade-off, we adopt the preserve strategy in our XAG evaluation framework.

2) Primary Inputs with Mixed Fanout: The encoding space assigned to each PI in an XAG depends on its fanout structure. If a PI drives only XOR gates, it can be encrypted directly in $\mathbb{B}$; If it drives only AND gates, encryption in $\mathbb{Z}_p$ suffices. However, when a PI fans out to both gate types – particularly to multiple AND trees with varying arities – multiple ciphertext variants may be required, each tailored to a different encoding space.

This creates a need for repeated ciphertext conversions to match the target encoding space of each consumer. By unifying the encoding space for all non-linear gates to a single $\mathbb{Z}_p$, this overhead is eliminated. Each PI then requires at most one switch PBS to convert its ciphertext between $\mathbb{B}$ and $\mathbb{Z}_p$, avoiding redundant conversions and simplifying the overall evaluation process.

3) XOR Gates with Mixed Fanout: When the output of an XOR gate feeds into multiple AND gates, each consumer may belong to a different AND tree with potentially distinct arities. In the general case, this would require replicating the XOR output across multiple encoding spaces to match the requirements of each AND tree, similar to the dilemma introduced in Section III-A2. This concern can also be addressed by homomorphically evaluating all AND trees in the same plaintext space, whose size p is determined by the arity of the largest AND tree. If so, a single switch PBS suffices to convert the XOR output from $\mathbb{B}$ to $\mathbb{Z}_p$, regardless of downstream fanout.

4) Strategic Evaluation Choices for XAGs: To summarize, the previous analyses (from Section III-A1 to III-A3) highlight two practical design choices that enable efficient TFHE circuit evaluation under a multi-encoding model:

a) AND tree consolidation: We treat consecutive two-input AND gates as a single logic unit – an $(n+1)$-input AND tree – which can be evaluated using a single PBS in a suitably large encoding space. This directly reduces the number of compute PBS operations compared to evaluating each AND gate individually. For instance, rather than evaluating n two-input ANDs using n PBS operations (each in $\mathbb{Z}_4$, see Fig. 1b), we evaluate the entire AND tree in one PBS

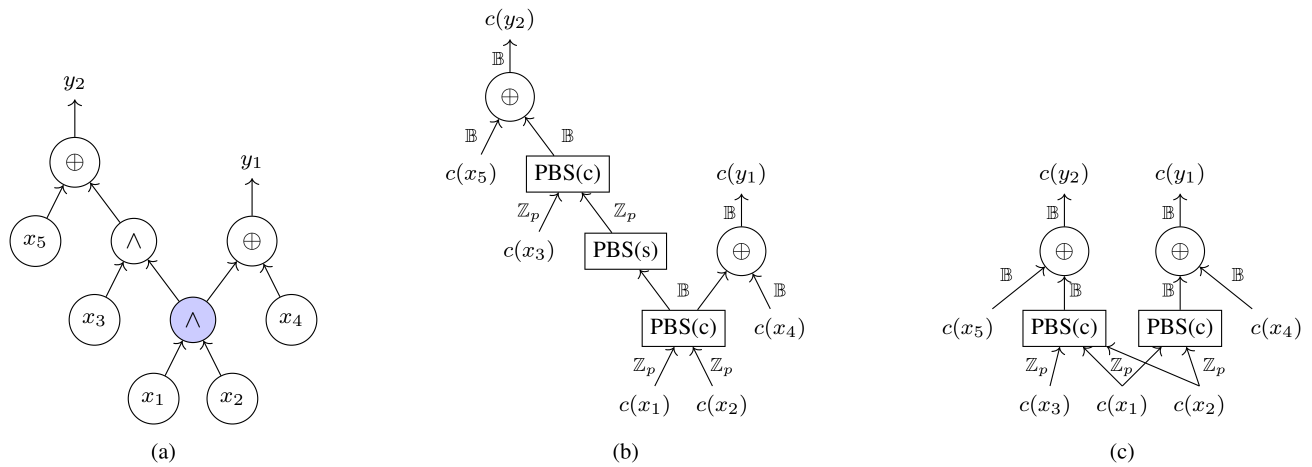


Fig. 3: Handling encoding transitions for an AND gate with mixed fanout. (a) A representative subnetwork in XAG form. The AND gate with mixed fanout is marked in blue. (b) Evaluation with AND tree splitting: introduces one switch PBS and two compute PBS operations. (c) Evaluation preserving the full AND tree: requires only two compute PBS operations, avoiding switch PBS.

using $\mathbb{Z}_{2(n+1)}$, which is sufficient to encode the required truth table (see Section II-A for details). This assumption improves evaluation efficiency without altering circuit semantics.

b) Uniform encoding for non-linear gates: Although different AND trees may have different arities and thus different ideal plaintext spaces, we unify the encoding space for all non-linear gates across the circuit. Specifically, we fix the encoding space to $\mathbb{Z}_p$, where $p = 2k_{\text{max}}$ and k_{max} is the arity of the largest AND tree in the XAG. This ensures that all AND trees, regardless of size, can be evaluated using a single compute PBS in $\mathbb{Z}_p$. While this may slightly over-provision smaller trees, it eliminates the need to duplicate ciphertexts across multiple spaces, thereby reducing the total number of switch PBS operations, as will be detailed later.

Together, these evaluation choices – motivated by the case studies above – align encoding strategy management with the structural properties of XAGs, minimizing total PBS usage and enabling efficient TFHE circuit evalation under the multi-encoding-space model.

B. Encoding Strategy Behavior in ESOP-Based Evaluation

The ESOP form of a Boolean function imposes a strict two-level logic structure that maximally separates non-linear and linear components. An ESOP can be interpreted as a constrained XAG, where the first level comprises only AND trees and the second level forms a single XOR tree that aggregates their outputs. This structural regularity makes ESOPs especially well-suited for multi-encoding-space TFHE evaluation, as it provides a clear boundary at which encoding transitions may occur.

1) Exclusive-or Sum of Products (ESOP) Form: The ESOP form expresses a Boolean function as the XOR of several product terms, each being a conjunction of literals (variables or their negations) [14]. Formally, for a function $f : \mathbb{B}^n \mapsto \mathbb{B}$, its ESOP form is given by:

$$f(x_1, x_2, \dots, x_n) = P_1 \oplus P_2 \oplus \cdots \oplus P_m,$$

where each P_i is a product term (cube) composed of one or more literals. For example, one valid ESOP form of the Boolean function implemented by the circuit in Fig. 2 is $f = x_1x_2 \oplus x_1x_3x_4$. It is important to note that the ESOP form of a Boolean function is not canonical; multiple structurally distinct ESOPs may represent the same function. ESOP minimization – finding a functionally equivalent expression with the fewest number of product terms – is known to be NP-hard, though many effective heuristic and exact synthesis methods have been developed [15]–[17].

2) Implications for Encoding Strategy Management: The ESOP structure enforces a clean separation between non-linear and linear operations, naturally aligning with distinct encoding domains: (1) All non-linear computation is confined to the first layer, where each product term (i.e., AND tree) is evaluated via a compute PBS in a suitable encoding space $\mathbb{Z}_p$. (2) The second layer aggregates the cube outputs using XOR operations, which are purely linear and can be evaluated in $\mathbb{B}$ via leveled operations (i.e., free-XOR) without invoking additional PBS operations.

To enable seamless aggregation between the two layers, we configure the output of each compute PBS (i.e., each product term) to adopt the plaintext space $\mathbb{B}$. This ensures that the XOR layer operates entirely within $\mathbb{B}$, avoiding the need for any switch PBS between the two layers.

3) Cost Model for ESOP Evaluation: We now formulate a quantitative cost model for evaluating a Boolean function represented in ESOP form. Let j_{pbs} denote the number of product terms with arity greater than one, each of which requires a compute PBS in the higher encoding space $\mathbb{Z}_p$. Let n_{pi} be the total number of PIs, and let m_{clean} be the number of inputs that can be evaluated entirely in a single encoding space – either because they appear only in higher-arity cubes or exclusively as arity-1 cubes. The number of switch PBS operations is then $(n_{\text{pi}} - m_{\text{clean}})$. Finally, let k_{max} denote the arity of the largest product term; the minimal required plaintext space is $p = 2k_{\text{max}}$ to encode its associated LUT.

To illustrate the use of this model, consider the ESOP representation of the Boolean function from Fig. 2, given by $f = x_1x_2 \oplus x_1x_3x_4$. Both product terms contain more than one literal and thus require compute PBS operations. Therefore, $j_{\text{pbs}} = 2$. The largest cube has arity three, so the minimal required plaintext space is $\mathbb{Z}_6$. As there are no arity-1 cubes, no switch PBS is required, yielding a switch PBS count of zero. In comparison, evaluating the original XAG implementation of the same function (also shown in Fig. 2) requires three PBS operations: two compute PBSs for gates n_5 and n_7, and one switch PBS to allow the output of gate n_6 to contribute to n_7. This example highlights the structural advantage of ESOP forms in minimizing encoding transitions and suggests their suitability for efficient multi-encoding-space TFHE circuit evaluation.

IV. ESOP-Guided TFHE Circuit Synthesis via Cost-Aware Cut Replacement

This section presents a synthesis methodology that transforms a Boolean circuit into an optimized network of ESOP-based subcircuits

Algorithm 1: Cut-Aware ESOP Cost Estimation

```
Input: Cut $C$: root $r$, leaf set $L$, cut function $f_C$
Output: Cost $c_{\text{local}}$
1  ESOP expression $\mathcal{E} \leftarrow \texttt{exorcism}(f_C)$
2  $j_{\text{pbs}} \leftarrow$ number of cubes in $\mathcal{E}$ with arity $> 1$
3  $m_{\text{clean}} \leftarrow 0$            // # of switch-free leaves
4  foreach leaf $l \in L$ do
5      if $l$ appears only in one cube of arity $1$ then
6          $m_{\text{clean}} \leftarrow m_{\text{clean}} + 1$
7  $n_{\text{pi}} \leftarrow |L|$
8  $s_{\text{switch}} \leftarrow n_{\text{pi}} - m_{\text{clean}}$
9  $c_{\text{local}} \leftarrow j_{\text{pbs}} + s_{\text{switch}}$
10 return $c_{\text{local}}$
```

for efficient TFHE evaluation. While Section III analyzed how to optimally evaluate a given circuit representation (XAGs or ESOPs) under the multi-encoding-space strategy, this section shifts attention to circuit generation. In particular, we explore how to synthesize improved implementations that unlock further efficiency gains, leveraging the structural and cost advantages of ESOP-based evaluation. To this end, we propose a cut-based mapping framework that decomposes the circuit into overlapping subgraphs and selectively replaces them with ESOP implementations. Each replacement is guided by a cost model that captures the trade-off between compute and switch PBS operations, enabling encoding-switch-aware optimization at fine granularity.

A. High-Level Flow

Given a Boolean circuit represented as a general logic network (e.g., an XAG), the synthesis proceeds as follows. For each node, we first enumerate all k-feasible cuts rooted at that node, each defining a subcircuit with at most k leaves and a self-contained Boolean function. For each cut, we derive an optimal ESOP representation using `exorcism` [16]. We then estimate the evaluation cost of each ESOP using a refined version of the cost model introduced in Section III-B3, which extends the original formulation to more accurately capture both compute and switch PBS operations during cut evaluation. Based on these costs, a heuristic selection algorithm identifies a set of non-overlapping cuts that cover the entire XAG and collectively minimize the total homomorphic evaluation cost. The resulting structure is a network of ESOP subcircuits, optimized for TFHE evaluation under an encoding-switch-aware strategy. This synthesis flow enables fine-grained trade-offs between local structural choices and global evaluation efficiency, producing modular circuits well-suited for multi-encoding-space TFHE evaluation.

B. Cut-Aware ESOP Cost Modeling

When applying ESOP-based evaluation to subcircuits (i.e., cuts) of a larger Boolean network, the cost model must be adapted to account for interactions between local ESOPs and their surrounding context. In particular, the inputs to a cut may originate from previously evaluated ESOP subgraphs. As discussed earlier, such outputs are encoded in the $\mathbb{B}$ plaintext space to enable free-XOR aggregation. If these ciphertexts are reused in product terms of arity greater than one (i.e., in the non-linear layer of a subsequent ESOP), they must be converted to a higher encoding space $\mathbb{Z}_p$ using a switch PBS.

To capture this scenario, we refine the cost model introduced in Section III-B3. The revised model, shown in Algorithm 1, estimates the total PBS cost of evaluating a given ESOP cut by accounting for both compute PBS operations (used to evaluate non-linear cubes) and switch PBS operations (used to align encoding spaces at the cut boundary).

Algorithm 2: Per-Round Cut Selection Procedure

```
Input: XAG $G$, global cost estimates $\texttt{best\_cost}(\cdot)$ from
       previous round
Output: Updated representative cut $\texttt{rep}(n)$ and cost
        $\texttt{best\_cost}(n)$ for each node $n \in G$
1  foreach node $n \in G$ in topological order do
2      foreach $k$-feasible cut $C$ rooted at $n$ do
3          $c_{\text{local}} \leftarrow \texttt{ESOP\_cost}(C)$        // Algorithm 1
4          $cost \leftarrow \texttt{cost\_est}(C, c_{\text{local}})$
5          if $cost < \texttt{best\_cost}(n)$ then
6              $\texttt{best\_cost}(n) \leftarrow cost$
7              $\texttt{rep}(n) \leftarrow C$
8  return $\texttt{rep}(n)$ and $\texttt{best\_cost}(n)$ for all $n$
```

This refined model provides a localized yet context-aware estimate of PBS cost, enabling accurate evaluation of cut replacements during synthesis.

C. Cost-Aware Cut Selection

To assemble a globally optimized TFHE circuit from locally evaluated ESOP cuts, we adopt a dynamic programming-based selection framework inspired by conventional technology mapping [18]. The objective is to select, for each node in the input XAG, a single representative cut whose ESOP-based implementation minimizes the overall homomorphic evaluation cost – accounting for both compute and switch PBS operations. This is achieved by iteratively refining the cut selection at each node.

As shown in Algorithm 2, for each node, all k-feasible cuts are evaluated using a modular cost estimator, `cost_est`, which integrates two components: (1) a local cost (‘c_{local}’), computed using the refined ESOP cost model from Algorithm 1, which accounts for the number of compute and switch PBS operations required to evaluate the cut; and (2) a global cost contribution (‘*cost*’), estimated based on how the cut affects the total cost of the circuit.

Estimating this global contribution precisely is non-trivial. To address this, the selection process proceeds over four iterations, alternating between two complementary heuristics for implementing `cost_est`: the first two rounds use the *area flow* heuristic [19] to capture global logic sharing, while the last two rounds use the *exact area* heuristic [20], which measures the size of a node's *maximum fanout-free cone* (MFFC) – defined as the largest subset of its transitive fan-in such that removing both the node and this subset does not affect the remaining circuit's functionality.

A cut is committed to a node if it yields a lower cost than the previously selected one. After the final iteration, the representative cuts (‘$\texttt{rep}(n)$’) define the mapping: the synthesized TFHE circuit is constructed by traversing the XAG in reverse topological order and instantiating the ESOP implementation of each selected cut.

This cost-aware selection framework enables fine-grained encoding-strategy optimization while preserving functional correctness, guiding synthesis toward TFHE circuits with minimal PBS overhead.

V. Experimental Results

This section evaluates the effectiveness of the proposed ESOP-guided synthesis technique for TFHE circuit generation on a suite of general-purpose Boolean benchmarks, assessing performance from two perspectives: (1) *Evaluation time comparison*, comparing homomorphic evaluation latency across synthesis approaches; and (2) *PBS type analysis*, breaking down *compute* and *switch* PBS operations to show the impact of encoding strategy management.

TABLE I: Comparison of TFHE circuit evaluation costs for general-purpose Boolean benchmarks.

Benchmark	*AND Minimization* [21]				$\mathbb{Z}_8$-*Based Synthesis* [11]			*This Work*			
	Max arity	#PBS	Syn. [s]	Eval. [ms]	#PBS	Syn. [s]	Eval. [ms]	Max arity	#PBS	Syn. [s]	Eval. [ms]
`cardio`	12	141	24.14	6 139	114	<0.01	4 560	8	120	5.98	4 392
`dsort`	5	789	564.00	29 316	663	<0.01	26 520	5	828	31.39	31 617
`msort`	5	690	50.37	23 715	645	<0.01	25 800	5	528	100.88	19 602
`isort`	5	690	50.17	23 715	645	<0.01	25 800	5	528	100.89	19 602
`bsort`	5	690	51.47	23 715	645	<0.01	25 800	5	528	100.64	19 602
`osort`	5	598	76.45	20 553	559	<0.01	22 360	5	460	83.38	17 058
`hd01`	32	64	0.43	5 096	54	<0.01	2 160	5	59	0.58	2 296
`hd02`	32	62	0.60	4 030	78	<0.01	3 120	5	81	1.33	2 791
`hd03`	4	51	0.44	1 679	29	<0.01	1 160	4	52	1.13	1 748
`hd04`	26	59	7.86	3 298	53	<0.01	2 120	7	65	2.28	2 500
`hd05`	17	191	20.70	9 173	113	<0.01	4 520	6	153	2.00	5 725
`hd06`	17	191	21.67	9 173	113	<0.01	4 520	6	153	2.02	5 725
`hd07`	4	15	0.05	555	16	<0.01	640	4	14	0.08	526
`hd08`	13	20	0.09	874	11	<0.01	440	8	12	0.17	508
`hd09`	33	69	1.02	4 521	76	<0.01	3 040	8	108	1.73	3 964
`hd10`	32	6	0.08	534	25	<0.01	1 000	8	16	0.27	624
`hd11`	48	168	8.37	23 397	310	<0.01	12 400	8	411	12.71	15 391
`hd12`	32	67	0.42	3 887	73	0.01	2 920	8	59	2.05	2 287
`bar`	3	1 980	25.37	61 887	2 437	<0.01	97 480	3	2 176	51.62	68 480
`cavlc`	30	365	19.12	22 300	518	<0.01	20 720	8	335	7.83	14 371
`ctrl`	10	61	1.85	2 605	84	<0.01	3 360	7	73	0.36	3 047
`dec`	8	280	25.37	12 280	292	<0.01	11 680	2	352	2.49	10 208
`i2c`	26	535	24.11	40 217	890	<0.01	35 600	8	703	10.07	29 171
`int2float`	14	127	5.91	6 665	162	<0.01	6 480	8	117	2.03	4 849
`router`	70	51	7.78	7 639	111	<0.01	4 440	8	95	1.51	3 555
GEOMEAN		141.32		7350.23	152.00		6 079.87		150.22		5 634.95
Norm.				**1**			**0.8272**				**0.7666**

A. Overview and Methodology

While prior work has demonstrated that hand-optimized cryptographic circuits, which feature clear separations between linear and non-linear components, can effectively leverage multi-encoding-space evaluation [6], such structural regularity is rarely found in broader application domains. As a result, the practicality of existing techniques remains limited when targeting general-purpose homomorphic evaluation of Boolean functions.

We therefore evaluate our synthesis framework using a diverse suite of general-purpose logic benchmarks. Specifically, we consider 25 circuits curated by prior work [22], spanning application areas such as medical diagnostics, sorting, bit-twiddling, random logic, and control logic. These circuits reflect the types of Boolean computations expected to arise in real-world FHE workloads. Unlike cryptographic circuits, these designs typically exhibit heterogeneous structure and do not feature deliberate separation between linear and non-linear logic. This makes them a representative and challenging testbed for evaluating synthesis strategies that aim to optimize homomorphic evaluation through encoding strategy management.

We compare against state-of-the-art synthesis strategies, including: (a) *AND minimization* [21], which aims to reduce the number of compute PBS operations by minimizing the number of AND gates in the circuit. This approach supports free XOR via dual-space evaluation; specifically, the encoding-space management strategy introduced in Section III-A for efficient evaluation. (b) $\mathbb{Z}_8$-*based synthesis* [11], which targets a fixed, larger encoding space ($\mathbb{Z}_8$) to reduce the number of PBS operations by enabling multi-output gates (via the multi-value PBS technique introduced in Section II-A). Since the encoding space remains constant, all PBS operations are of the compute type.

We implemented the proposed synthesis framework atop the C++ logic synthesis library `mockturtle` [2] [23]. The homomorphic evaluation latency was estimated using a TFHE cost estimator [3], which integrates with the `Concrete` compiler [24] for accurate per-PBS-operation timing. For approaches that support free-XOR, the cost of the leveled operations used to realize XOR gates is omitted, as it is negligible compared to the cost of PBS operations. The experiment was conducted on a machine equipped with an Apple M1 Max processor and 32GB of memory.

B. Performance Breakdown

We begin by comparing the total homomorphic evaluation latency of circuits synthesized by three approaches. For each benchmark, we report: the arity of the largest AND tree in the synthesized circuit ('max arity'), the total number of PBS operations required for evaluation ('#PBS'), including both compute and switch PBS where applicable; the circuit synthesis time in seconds ('Syn.'), and the total homomorphic evaluation time in milliseconds ('Eval.').

a) Comparison of evaluation time: A key observation is that no single approach dominates all 25 benchmarks. The best evaluation time is achieved by AND minimization, $\mathbb{Z}_8$-based synthesis, and ESOP-guided synthesis for 2, 9, and 14 benchmarks, respectively. This indicates that the three approaches exhibit complementary strengths and exploit different structural properties of the benchmarks.

Notably, $\mathbb{Z}_8$-based synthesis consistently outperforms others on most benchmarks that perform bit-twiddling operations (i.e., `hd-01` to `hd-12`). This aligns with its core design principle: leveraging specialized large gates (e.g., symmetric or negacyclic functions) that can be efficiently implemented in $\mathbb{Z}_8$ despite the constraints imposed by the ring's negacyclic structure. Additionally, its support of multi-output gates facilitated by the multi-value PBS technique, as introduced in Section II-A, further improves circuit compactness.

In contrast, the ESOP-guided synthesis generally achieves the best performance on the remaining benchmarks, which span diverse applications such as medical diagnostics and control logic. This

[2]Available at https://github.com/lsils/mockturtle

[3]Available at https://github.com/ssmiler/tfhe_lbf_eval

reinforces the broader applicability of our method in scenarios where circuit structure is less regular or less conducive to domain-specific optimization. It also suggests a promising direction for further research on categorizing Boolean functions by their compatibility with different TFHE circuits synthesis strategies.

b) Circuit synthesis time: Among all methods, $\mathbb{Z}_8$-based synthesis achieves the fastest synthesis time due to its technology mapping approach. Both AND minimization and ESOP-guided synthesis exhibit comparable runtimes, generally on the order of minutes.

The higher runtime of AND minimization arises from its iterative application of multiple high-effort optimization passes, which converge slowly on structurally complex circuits. Meanwhile, ESOP-guided synthesis incurs runtime overhead from dynamic per-cut ESOP synthesis, which dominates the runtime. This limitation could be alleviated by precomputing a database of optimized ESOP representations to avoid on-the-fly and potentially redundant synthesis.

It is important to emphasize that, in many use cases – especially those where homomorphic evaluation is deployed as a secure cloud service – the evaluation time, rather than synthesis time, is the dominant concern. Circuit generation can be amortized and reused across multiple runs, making offline synthesis effort more tolerable.

c) Relationship between PBS count and evaluation time: A counterintuitive observation from Table I is that, although circuits synthesized via AND minimization exhibit the lowest average PBS count, they also result in the highest average evaluation latency.

This is explained by the 'max arity' column. We observe that AND minimization often produces circuits containing large AND trees, which increases the arity of the non-linear components. To evaluate such trees via a single PBS, the corresponding LUT must be encoded in a larger plaintext space $\mathbb{Z}_p$, resulting in higher per-PBS cost. In contrast, ESOP-guided synthesis imposes more uniform boundaries between linear and non-linear computation. By avoiding product terms with excessive literal counts, it ensures that PBS operations can be performed using moderate-sized encoding spaces, thereby reducing overall evaluation cost despite slightly higher PBS counts.

This trade-off illustrates the effectiveness of the proposed ESOP-based strategy: prioritizing manageable PBS complexity via structural control leads to more efficient homomorphic evaluation in practice.

C. Analysis of PBS Types

To understand the cost implications of encoding management, we analyze the number of switch PBS and compute PBS operations involved in evaluating circuits synthesized by the AND minimization flow and our proposed ESOP-guided synthesis method. Since $\mathbb{Z}_8$-based synthesis does not support encoding switching, only compute PBS operations are involved and thus excluded from this part of the comparison.

By breaking down the total PBS count ('#PBS') reported in Table I into switch PBS ('#Switch') and compute PBS ('#Compute') components, as detailed in Table II, we observe that the proposed ESOP-guided synthesis approach achieves an average 23.62% reduction in switch PBS count compared to the AND minimization baseline. This result highlights the advantage of our method in managing encoding transitions, which is the central goal of this work.

This improvement comes at the cost of a 35.48% increase in compute PBS count, which is expected given that AND minimization directly targets reducing the number of AND gates in XAGs. Since the homomorphic evaluation of AND gates corresponds to compute PBS operations, aggressive AND reduction naturally lowers the compute PBS count. However, as discussed in Section V-B, the lack of encoding-aware synthesis in the AND minimization flow leads not only to more switch PBS operations but also to the formation of excessively large AND trees. This increases the required plaintext space size and, consequently, the cost per compute PBS. In contrast, the ESOP-guided synthesis approach inherently avoids this issue by producing structurally uniform circuits with well-bounded AND tree arities, enabling more efficient parameter selection and lower per-PBS evaluation cost.

TABLE II: Comparison of usage of PBS operations.

Benchmark	*AND Minimization* [21]		*This Work*	
	#Switch	#Compute	#Switch	#Compute
`cardio`	91	50	63	57
`dsort`	294	495	243	585
`msort`	405	285	198	330
`isort`	405	285	198	330
`bsort`	405	285	198	330
`osort`	351	247	174	286
`hd01`	19	45	14	45
`hd02`	31	31	47	34
`hd03`	31	20	28	24
`hd04`	36	23	24	41
`hd05`	112	79	61	92
`hd06`	112	79	61	92
`hd07`	3	12	2	12
`hd08`	13	7	2	10
`hd09`	34	35	56	52
`hd10`	1	5	6	10
`hd11`	73	95	194	217
`hd12`	40	27	23	36
`bar`	491	1 489	384	1 792
`cavlc`	200	165	44	291
`ctrl`	23	38	11	62
`dec`	20	260	48	304
`i2c`	177	358	154	549
`int2float`	56	71	26	91
`router`	31	20	45	50
GEOMEAN	59.93	71.92	45.78	97.44
Norm.	**1**	**1**	**0.7638**	**1.3548**

Consequently, while both the proposed ESOP-guided synthesis and the AND minimization baseline support multi-encoding evaluation, only the former achieves a systematic management of encoding transitions through synthesis, resulting in better TFHE circuit designs.

VI. Conclusion

TFHE circuit synthesis aims to generate Boolean circuits for homomorphic evaluation, where each gate is evaluated via a *programmable bootstrapping* (PBS) operation. Mainstream approaches assume a fixed plaintext space for all gates, missing optimization opportunities from dynamic encoding strategies. Recent advances enable encoding space transitions via additional PBS operations, allowing linear gates in $\mathbb{B}$ to be evaluated "for free" using the free-XOR technique. Fully exploiting this capability requires explicit *encoding strategy management* to avoid excessive or poorly placed transitions. This work introduces the first systematic framework for *encoding-switch-aware* TFHE circuit synthesis, combining structural analysis with cost-aware ESOP-based mapping to manage encoding transitions effectively. Across 25 general-purpose benchmarks, our method reduces homomorphic evaluation time by up to 53.46% and 23.34% on average over advanced synthesis baselines without explicit encoding-switch management. Future work includes constraining AND tree arity to limit PBS complexity and combining exact and heuristic ESOP synthesis to improve subcircuit quality without excessive runtime.

Acknowledgment

The authors would like to thank the anonymous reviewers for their valuable feedback and Dr. Sergiu Carpov for the fruitful discussion. This work was supported in part by Synopsys Inc.

References

[1] J. Fan and F. Vercauteren, “Somewhat practical fully homomorphic encryption,” Cryptology ePrint Archive, Paper 2012/144, 2012.

[2] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, “(Leveled) fully homomorphic encryption without bootstrapping,” in *Innovations in Theoretical Computer Science*. ACM, 2012, pp. 309–325.

[3] L. Ducas and D. Micciancio, “FHEW: Bootstrapping homomorphic encryption in less than a second,” in *EUROCRYPT*, 2015, pp. 617–640.

[4] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, “TFHE: Fast fully homomorphic encryption over the torus,” *Journal of Cryptology*, vol. 33, pp. 34–91, 2020.

[5] J. H. Cheon, A. Kim, M. Kim, and Y. Song, “Homomorphic encryption for arithmetic of approximate numbers,” in *Advances in Cryptology – ASIACRYPT 2017*, 2017, pp. 409–437.

[6] N. Bon, D. Pointcheval, and M. Rivain, “Optimized homomorphic evaluation of boolean functions,” *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2024, no. 3, pp. 302–341, 2024.

[7] L. Bergerat, A. Boudi, Q. Bourgerie, I. Chillotti, D. Ligier, J.-B. Orfila, and S. Tap, “Parameter optimization and larger precision for (T)FHE,” *Journal of Cryptology*, vol. 36, no. 28, 2023.

[8] L. Bergerat, I. Chillotti, D. Ligier, J.-B. Orfila, and S. Tap, “TFHE gets real: an efficient and flexible homomorphic floating-point arithmetic,” *IACR Transactions on Cryptographic Hardware and Embedded Systems*, vol. 2025, no. 2, pp. 126–162, 2025.

[9] Z. Guan, R. Mao, Q. Zhang, Z. Zhang, Z. Zhao, and S. Bian, “AutoHoG: Automating homomorphic gate design for large-scale logic circuit evaluation,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 43, no. 7, pp. 1971–1983, 2024.

[10] S. Carpov, “A fast heuristic for mapping Boolean circuits to functional bootstrapping,” Cryptology ePrint Archive, Paper 2024/1204, 2024.

[11] M. Yu, S. Carpov, A. Tempia Calvino, and G. De Micheli, “On the synthesis of high-performance homomorphic Boolean circuits,” in *Proceedings of the 12th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, 2024, pp. 51–63.

[12] S. Carpov, M. Izabachène, and V. Mollimard, “New techniques for multi-value input homomorphic evaluation and applications,” in *Topics in Cryptology – CT-RSA 2019: The Cryptographers’ Track at the RSA Conference*, 2019, pp. 106–126.

[13] I. Chillotti, D. Ligier, J.-B. Orfila, and S. Tap, “Improved programmable bootstrapping with larger precision and efficient arithmetic circuits for TFHE,” in *Advances in Cryptology – ASIACRYPT 2021: 27th International Conference on the Theory and Application of Cryptology and Information Security*, 2021, pp. 670–699.

[14] I. Reed, “A class of multiple-error-correcting codes and the decoding scheme,” *Transactions of the IRE Professional Group on Information Theory*, vol. 4, no. 4, pp. 38–49, 1954.

[15] M. Perkowski and M. Chrzanowska-Jeske, “An exact algorithm to minimize mixed-radix exclusive sums of products for incompletely specified Boolean functions,” in *IEEE International Symposium on Circuits and Systems*, 1990, pp. 1652–1655.

[16] A. Mishchenko and M. Perkowski, “Fast heuristic minimization of exclusive-sums-of-products,” in *5th International Reed-Muller Workshop*, 2001.

[17] H. Riener, R. Ehlers, B. d. O. Schmitt, and G. De Micheli, “Exact synthesis of ESOP forms,” in *Advanced Boolean Techniques*, 2020, pp. 177–194.

[18] A. Tempia Calvino, H. Riener, S. Rai, A. Kumar, and G. De Micheli, “A versatile mapping approach for technology mapping and graph optimization,” in *Proceedings of the 27th Asia and South Pacific Design Automation Conference*, 2022, pp. 410–416.

[19] V. Manohararajah, S. D. Brown, and Z. G. Vranesic, “Heuristics for area minimization in LUT-based FPGA technology mapping,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 11, pp. 2331–2340, 2006.

[20] A. Mishchenko, S. Cho, S. Chatterjee, and R. Brayton, “Combinational and sequential mapping with priority cuts,” in *IEEE/ACM International Conference on Computer-Aided Design*, 2007, pp. 354–361.

[21] E. Testa, M. Soeken, H. Riener, L. Amaru, and G. De Micheli, “A logic synthesis toolbox for reducing the multiplicative complexity in logic networks,” in *Design, Automation & Test in Europe Conference & Exhibition*, 2020, pp. 568–573.

[22] D. Lee, W. Lee, H. Oh, and K. Yi, “Optimizing homomorphic evaluation circuits by program synthesis and term rewriting,” in *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2020, pp. 503–518.

[23] M. Soeken, H. Riener, W. Haaswijk, E. Testa, B. Schmitt, G. Meuli, F. Mozafari, S.-Y. Lee, A. Tempia Calvino, D. S. Marakkalage, and G. De Micheli, “The EPFL logic synthesis libraries,” arXiv 1805.05121, 2022.

[24] Zama, “Concrete: TFHE compiler that converts python programs into FHE equivalent,” 2022, https://github.com/zama-ai/concrete.