Beyond the Bit: Key Innovations in TFHE

Hussein Aghajani Kalkhouran

Department of Computer Science, AGH University of Science and Technology

Krakow, Poland

Abstract

Fully Homomorphic Encryption (FHE), particularly TFHE, offers significant promise for privacy-preserving computation but continues to face practical challenges related to performance, scalability, and real-world integration. This in-depth review synthesizes six particularly impactful and representative papers published between 2024 and 2025. These works were selected for their novel contributions across algebraic foundations, performance enhancements, and practical applications of TFHE, collectively demonstrating substantial progress toward maturing the technology. We highlight TFHE's evolving efficiency, applicability, and security by examining advancements in bootstrapping techniques, algebraic innovations, and system-level integrations. Significant advancements in TFHE-like bootstrapping are evident through multiple innovations. Novel GSW configurations demonstrate substantial performance gains, LWE modulus, pushing bootstrappable precision beyond 32 bits with remarkable speedups exceeding 10 × and key size reductions approaching three orders of magnitude at 11-bit precision. Large-plaintext functional bootstrapping techniques based on monic monomial permutation matrices (MMPM) further enhance both key sizes and runtime efficiency. TFHE's real-world utility is also expanding across critical application domains. In federated learning, blockchain-integrated TFHE frameworks enhance robustness against poisoning attacks, achieving superior accuracy retention and reducing encryption and decryption overhead by up to 28.4%. In genomics, TFHE-style circuits enable privacy-preserving Banded Smith–Waterman alignment, with SIMD-optimized implementations demonstrating competitive amortized performance. In cloud computing, hybrid FHE architectures leverage TFHE for integrity-critical operations and comparisons, integrating verifiability mechanisms via homomorphic message authentication codes and sophisticated cost models that enable elastic, sub-second confidential computation. Collectively, these works demonstrate a strong and coherent push toward transforming TFHE into a robust and scalable foundation for privacy-preserving computation. This review provides a critical synthesis of the algebraic innovations, system-level optimizations, and hybrid cryptographic designs driving TFHE's evolution, while also identifying remaining challenges and promising directions for future research.

Keywords:

Fully Homomorphic Encryption (FHE), TFHE, Bootstrapping, Confidential Computing, Federated Learning, Genomics, Algebraic Cryptography, Privacy-Preserving AI, Blockchain.

1. Introduction

The pervasive digitization of information and the proliferation of cloud computing, artificial intelligence, and the Internet of Things have ushered in an era of unprecedented data generation and processing capabilities. While these advancements promise transformative societal benefits, they simultaneously amplify critical concerns regarding data privacy and confidentiality. Sensitive information, ranging from personal health records and financial transactions to proprietary business intelligence, is increasingly entrusted to third-party services, necessitating robust mechanisms to protect it from unauthorized access, misuse, or leakage.

Fully Homomorphic Encryption (FHE) emerges as a cryptographic panacea in this landscape, offering the revolutionary ability to perform arbitrary computations directly on encrypted data, yielding results that, upon decryption, are identical to those obtained from operations on plaintext. This paradigm-shifting capability promises to decouple trust from utility, enabling the full exploitation of outsourced computational resources without compromising the underlying data's privacy. The journey of FHE, initiated by Gentry's seminal work in 2009 [7], has been marked by continuous innovation aimed at bridging the formidable gap between theoretical feasibility and practical deployability. Early FHE schemes, while conceptually profound, were plagued by prohibitive computational overhead and rapid noise accumulation, rendering them impractical for real-world applications. Subsequent generations introduced significant optimizations, such as leveled FHE, Ring-Learning-With-Errors (RLWE) based constructions, and efficient noise management techniques like modulus switching. Among these advancements, the FHEW and TFHE (Torus FHE) schemes have carved out a distinct and highly impactful niche, primarily due to their groundbreaking approach to bootstrapping [12]. TFHE, building upon the foundations laid by FHEW, revolutionized FHE by introducing fast, gate-by-gate

Manuscript received April 5, 2026
Manuscript revised April 20, 2026
https://doi.org/**10.22937/IJCSNS.2026.26.4.1**

bootstrapping [12]. Unlike other FHE families that rely on polynomial interpolation or approximation for modular arithmetic, TFHE's ability to refresh noise on individual encrypted bits or small integers with millisecond-level latency fundamentally altered the FHE landscape. This granular control over noise management enables the direct evaluation of Boolean circuits and complex functions via lookup tables, making it particularly attractive for scenarios demanding low-latency, arbitrary-depth computations, such as comparisons, conditional logic, and privacy-preserving machine learning inference. Despite these remarkable strides, the widespread adoption of TFHE in production environments continues to face formidable challenges. These include the inherent performance overhead of ciphertext arithmetic compared to plaintext operations, the complexity of parameter selection to balance security, correctness, and efficiency, the substantial memory footprint of bootstrapping keys, and the intricate task of integrating TFHE into existing system architectures. The "rigid coupling" between cryptographic parameters, such as the polynomial dimension and the plaintext modulus, often constrains scalability and limits the achievable precision of encrypted computations [5].

Furthermore, the sheer novelty and complexity of FHE often impede its seamless integration into cloud computing paradigms, which demand elasticity, verifiability, and cost-effectiveness. This review paper aims to provide a focused and detailed synthesis of cutting-edge research in TFHE, drawing insights from six recent and impactful publications [1, 2, 3, 4, 5, 6] . These papers were selected for their significant and diverse contributions to advancing TFHE's practical utility, encompassing fundamental algebraic innovations, substantial performance and precision enhancements, and pioneering application-specific integrations. They collectively represent the forefront of TFHE development between 2024 and 2025, offering a holistic view of the field's current trajectory. Our objective is to critically examine the latest advancements that address the aforementioned challenges, highlighting the innovative algebraic constructions, system-level optimizations, and hybrid cryptographic approaches that are propelling TFHE towards broader practical utility. By dissecting these contemporary works, we seek to elucidate the current state of TFHE's efficiency, its expanding applicability across diverse domains, and the architectural considerations necessary for its robust deployment, while also identifying remaining limitations and future research avenues. Specifically, this review will delve into several key themes: 1. Algebraic Innovations: We will explore novel algebraic structures, such as new GSW configurations [1] and the groundbreaking RN-module framework [5], that fundamentally enhance the efficiency and scalability of TFHE-like bootstrapping by decoupling critical cryptographic parameters. This includes examining the use of monic monomial permutation matrices (MMPM) for large-plaintext functional bootstrapping [2]. 2. Performance and Precision Enhancements: We will analyze how these algebraic advancements translate into tangible performance gains, including significant speedups, drastic reductions in bootstrapping key sizes, and an unprecedented expansion of bootstrappable message precision [1, 2, 5] . 3. Application-Specific Optimizations: The review will investigate TFHE's tailored integration into critical application domains, including its role in securing federated learning frameworks against poisoning attacks [4] and its contribution to privacy-preserving genomics computations [3]. 4.

System-Level Integration and Verifiability: We will explore architectural blueprints for deploying FHE, including TFHE components, in cloud environments, focusing on strategies for achieving elastic, verifiable, and confidential compute [6]. This includes discussions on hybrid FHE schemes, homomorphic Message Authentication Codes (MACs) for verifiability, and sophisticated cost modeling for resource scheduling. By synthesizing these diverse contributions, this review aims to offer a holistic perspective on TFHE's rapid evolution. It will serve as a valuable resource for researchers, developers, and practitioners seeking to understand the current capabilities, ongoing challenges, and future directions of TFHE, ultimately charting a path towards its pervasive adoption in the increasingly privacy-conscious digital world. The remainder of this paper is organized as follows: Section 2 provides a brief overview of the core concepts of FHE and TFHE. Section 3 presents a detailed analysis of the algebraic innovations driving TFHE's performance [1, 2, 5]. Section 4 discusses system-level optimizations and architectural considerations for TFHE deployment [3, 4, 6]. Section 5 explores TFHE's application in federated learning and genomics [3, 4]. Finally, Section 6 concludes with a summary of findings and outlines future research directions.

Fully Homomorphic Encryption (FHE) is an encryption technique that enables unlimited computations over encrypted data. Over the past decade, FHE schemes have garnered significant attention due to their potential to solve real-world applications while preserving the privacy of manipulated data. The current focus is on FHE schemes based on hard lattice problems, specifically the LWE [9] and its variants RLWE [10]. Various techniques have been developed, including GSW [13], FHEW [12], TFHE [14, 15], HEAAN [16],

and BFV/BGV [17]. One of the primary challenges in fully homomorphic encryption (FHE) is to identify secure and efficient parameters regarding computational cost and memory usage. This is crucial for the large-scale adoption of FHE schemes. While the LWE/Lattice-estimator is used to evaluate the security of parameters in an LWE-based cryptosystem, it does not help to identify efficient parameters for a particular context. Finding the optimal parameter set is even more complex, and no solution has been proposed.

The idea of using the bootstrap procedure for table lookup to evaluate an arbitrary function was later developed into the Functional Bootstrap approach, which was strengthened by the faster bootstraps of TFHE. However, there has been little effort to define efficient methods for using this approach to implement high-precision functions. Regev introduced the Learning With Errors (LWE) problem in 2005 [9], while its Ring variant, RingLWE, was later proposed by Lyubashevsky, Peikert, and Regev in 2010 [10]. Both variants are currently widely utilized for constructing lattice-based Homomorphic Encryption schemes. The original LWE definition [11] involves the right-hand side of an LWE sample being on the torus and defined using continuous Gaussian distributions. In this context, the authors work solely on the real torus, employing the same formalism as the Scale Invariant LWE scheme in [13] or the LWE scale-invariant standard form in [8], referred to as LWE without loss of generality.

2 Preliminaries: Foundations of Fully Homomorphic Encryption and TFHE

To fully appreciate the innovations and advancements in TFHE discussed in this review, a solid understanding of the foundational principles of Fully Homomorphic Encryption (FHE) and the specific architectural choices of TFHE is essential. This section lays out the necessary theoretical and practical groundwork, defining key concepts, mathematical underpinnings, and operational mechanisms that characterize FHE in general and TFHE in particular.

2.1 General Concepts of Fully Homomorphic Encryption (FHE)

Fully Homomorphic Encryption (FHE) is a cryptographic primitive that enables arbitrary computations on encrypted data without prior decryption. This revolutionary capability allows sensitive data to remain confidential even when processed by untrusted third parties, such as cloud service providers.

2.1.1 Definition and Properties

Formally, a Fully Homomorphic Encryption (FHE) scheme consists of four algorithms: *KeyGen*, *Encrypt*, *Decrypt*, and *Evaluate*. The core correctness property is that for any function f and any messages $m_1, m_2, \dots, m_k$,

$$\text{Decrypt}\big(\text{SK}, \text{Evaluate}(\text{EK}, \text{Encrypt}(\text{PK}, m_1), \dots, \text{Encrypt}(\text{PK}, m_k))\big) = f(m_1, \dots, m_k),$$

where PK is the public key, SK is the secret key, and EK is the evaluation key.

FHE schemes are typically classified into the following categories:

- **Partially Homomorphic Encryption (PHE):** Supports an unlimited number of only one type of operation, such as addition in Paillier encryption or multiplication in RSA.
- **Somewhat Homomorphic Encryption (SWHE):** Supports both addition and multiplication, but only up to a bounded circuit depth, after which noise accumulation prevents correct decryption. Many early FHE constructions were initially developed as SWHE schemes.
- **Fully Homomorphic Encryption (FHE):** Supports an unlimited number of both addition and multiplication operations, enabling the evaluation of circuits of arbitrary depth. This capability is achieved through a noise-refreshing mechanism known as *bootstrapping* [7].

Key properties of FHE schemes include:

- **Semantic Security:** An adversary cannot distinguish between encryptions of different plaintexts, even under chosen-plaintext attacks and with access to the public key. This property is typically ensured by injecting random noise during encryption.
- **Probabilistic Encryption:** Encrypting the same plaintext multiple times yields distinct ciphertexts, further strengthening semantic security.
- **Malleability:** FHE schemes are inherently malleable. Given a ciphertext $C = \text{Encrypt}(\text{PK}, m)$, it is possible to derive another

ciphertext C' that decrypts to a related plaintext m'. This malleability is precisely the property exploited to enable homomorphic computation.

- **Compactness:** The size of ciphertexts remains independent of the number of homomorphic operations performed, ensuring that the complexity of the decryption algorithm does not grow with the evaluated circuit.

2.1.2 Noise Management

A fundamental challenge in FHE is the management of "noise" (or error) inherent in the ciphertexts. For security, FHE schemes typically add a small random error component during the encryption process. This noise ensures that the ciphertext appears random, even if the plaintext is known. However, homomorphic operations, particularly multiplication, cause this noise to grow. If the accumulated noise exceeds a certain threshold, the ciphertext can no longer be correctly decrypted, leading to computational failure. This phenomenon defines the "noise budget" of a ciphertext – the maximum amount of noise it can tolerate before becoming undecipherable. The rate of noise growth is a critical metric for FHE scheme efficiency.

2.1.3 Bootstrapping

The core idea is to homomorphically evaluate the scheme's own decryption circuit. Given a noisy ciphertext C and an encrypted secret key Encrypt(PK,SK), the Evaluate algorithm computes the decryption function on these inputs. The output is a new ciphertext C' that encrypts the same plaintext as C but with a significantly reduced noise level. This process, often called *Recrypt*, effectively "cleans" the ciphertext, allowing for an unlimited number of subsequent homomorphic operations. Historically, bootstrapping was extremely computationally expensive, taking minutes for a single bit operation [7]. Much of the subsequent FHE research has focused on accelerating this crucial operation.

2.1.4 Leveled FHE

For applications requiring a bounded number of homomorphic operations, Leveled FHE offers a practical alternative to bootstrapping. Instead of refreshing noise, leveled FHE schemes are parameterized such that the initial noise budget is large enough to accommodate the noise growth throughout the entire computation up to a predefined multiplicative depth. This approach involves selecting appropriate parameters (e.g., polynomial degree, ciphertext moduli) based on the maximum required circuit depth, desired security level, and acceptable decryption failure rate. While it doesn't offer the arbitrary depth of true FHE, leveled FHE often provides better performance for fixed-depth circuits by avoiding the overhead of bootstrapping. Many practical FHE applications today utilize leveled FHE.

2.1.5 Underlying Hardness Assumptions

Modern FHE schemes derive their security from the presumed hardness of certain mathematical problems, primarily those related to lattices. These problems are believed to be resistant to attacks from both classical and quantum computers, making FHE a candidate for post-quantum cryptography.
Learning With Errors (LWE): Introduced by Regev [9], the LWE problem states that it is hard to distinguish between truly random samples and samples of the form

$$(\boldsymbol{a},\ \boldsymbol{a}\cdot\boldsymbol{s}+\boldsymbol{e}\ (\mathrm{mod}\ q))$$

where $\boldsymbol{a}$ is a random vector, $\boldsymbol{s}$ is a small secret vector, $\boldsymbol{e}$ is a small error vector (noise), and q is a modulus. The security relies on the "smallness" of the secret and error.
Ring-LWE (RLWE): A more efficient variant of LWE, RLWE [10] extends the problem to polynomial rings $R=\mathbb{Z}[X]/(X^N+1)$. Samples are of the form

$$(A(X),\ A(X)\cdot S(X)+E(X)\ \mathrm{mod}\ Q(X))$$

where $A(X), S(X), E(X)$ are polynomials in $R_Q = R/QR$. RLWE-based schemes benefit from the algebraic structure of polynomial rings, enabling faster operations via the Number Theoretic Transform (NTT) or Fast Fourier Transform (FFT).
NTRU-Based Assumptions: Some FHE schemes, particularly earlier constructions or schemes tailored to specific algebraic structures, are based on the NTRU hardness assumption [11]. Wang et al. [1] discuss NTRU-based FHE in the context of bootstrapping optimizations, while Yadavalli et al. [6] explicitly investigate NTRU-based bootstrapping within a cloud-native FHE architecture.

2.2 Introduction to TFHE and its FHEW Lineage

TFHE (Fully Homomorphic Encryption over Torus) and its predecessor FHEW (FHE over the Weighed-sum) represent a significant leap in FHE efficiency, particularly for low-latency, gate-by-gate operations. Their design is optimized for fast bootstrapping, enabling complex computations on encrypted bits or small integers.

2.2.1 Torus Arithmetic

A defining characteristic of TFHE is its use of the torus $\mathbb{T} = \mathbb{R}/\mathbb{Z}$ as the plaintext and noise space. The torus can be visualized as a circle of circumference 1, where numbers are represented by their fractional part. For instance, $0.75 \in \mathbb{T}$ could represent a plaintext message.

In practical implementations, the continuous torus is discretized by scaling values by a large integer Q' (commonly 2^{32} or 2^{64}) and performing arithmetic modulo Q'. This effectively maps $\mathbb{T}$ to $\mathbb{Z}_{Q'}$, allowing efficient integer arithmetic on fixed-point numbers. Noise is naturally represented as small deviations on the torus, and decryption involves rounding to the nearest multiple of the plaintext modulus (e.g., 0 or 0.5 for binary messages).

2.2.2 Ciphertext Structure

TFHE employs two primary ciphertext types, both derived from the LWE problem:

TLWE (Torus LWE): This is the basic ciphertext format for encrypting a single bit or a small integer $\mu \in \mathbb{T}$. A TLWE ciphertext is a pair $(\boldsymbol{a}, b) \in \mathbb{T}^n \times \mathbb{T}$ (or often as $(\boldsymbol{a}, b) \in \mathbb{Z}_q^n \times \mathbb{T}$ where $\boldsymbol{a}$ is sampled from $\mathbb{Z}_q$). The encryption of a plaintext μ under a secret key $\boldsymbol{s} \in \mathbb{Z}_q^n$ is given by:

$$b \approx \langle \boldsymbol{a}, \boldsymbol{s} \rangle + \mu + e \pmod 1$$

where $\boldsymbol{a}$ is a random vector, and $e \in \mathbb{T}$ is a small Gaussian noise term. Decryption involves computing the "phase" $b - \langle \boldsymbol{a}, \boldsymbol{s} \rangle \pmod 1$ and rounding it to the nearest plaintext value.

TGSW (Torus GSW): Based on the GSW scheme [13], a TGSW ciphertext encrypts a small integer or bit $m \in \mathbb{T}$ into a matrix of TLWE ciphertexts. Its structure is more complex, typically represented as an $(n+1) \times (L \cdot B)$ matrix of elements from $\mathbb{T}$, where L is the gadget decomposition length and B is the base. The primary purpose of TGSW ciphertexts is to enable homomorphic multiplication and to control other ciphertexts, particularly in the External Product and CMux operations. The "gadget matrix" [1] (Section 3) is central to its construction, allowing for efficient decomposition and recombination during operations.

2.2.3 Key Components and Operations

TFHE's efficiency stems from a set of highly optimized homomorphic operations:

Gadget Decomposition: This technique is fundamental to TGSW operations. It involves representing a large integer a as a sum of its digits in a chosen base B:

$$a = \sum_{j=0}^{\ell-1} a_j B^j, \qquad a_j \in [-B/2, B/2)$$

This decomposition allows complex operations on large numbers to be broken down into simpler operations on their digits, reducing computational complexity and managing noise growth. Wang et al. [1] refer to this as GadgetDecomp.

External Product (TGSW ⊡ TLWE): This is the homomorphic multiplication operation in TFHE. It takes a TGSW ciphertext encrypting m_1 and a TLWE ciphertext encrypting m_2, and produces a new TLWE ciphertext encrypting $m_1 \cdot m_2$. The operation involves decomposing the TLWE ciphertext using the gadget decomposition and then performing a series of additions and scalar multiplications of TLWE ciphertexts, guided by the TGSW matrix structure:

$$C_{\text{TGSW}}(m_1) \boxdot C_{\text{TLWE}}(m_2) \rightarrow C_{\text{TLWE}}(m_1 \cdot m_2)$$

CMux (Controlled Multiplexer) Gate: The CMux gate is a cornerstone of TFHE, enabling conditional logic on encrypted data. It takes a TGSW ciphertext encrypting a control bit b, and two TLWE ciphertexts C_0 and C_1. It homomorphically outputs C_1 if $b = 1$ and C_0 if $b = 0$. The operation is defined as:

$$\begin{aligned}&\text{CMux}(C_{\text{TGSW}}(b), C_0, C_1)\\ &= C_{\text{TGSW}}(b) \boxdot (C_1 - C_0) + C_0\end{aligned}$$

This allows for the construction of arbitrary Boolean circuits and is a critical component of TFHE's fast bootstrapping.

2.2.4 TFHE Bootstrapping (Gate-by-Gate)

TFHE's most celebrated feature is its fast, gate-by-gate bootstrapping mechanism, which can refresh noise in milliseconds . This speed is achieved through a technique known as *blind rotation* and the use of *test vectors* or *test polynomials*.

- **Blind Rotation:** The core of TFHE bootstrapping. It homomorphically evaluates the *phase* of a noisy TLWE ciphertext. This process involves a series of CMux operations, controlled by components of the encrypted secret key. The result is a polynomial (or vector) whose constant term (or first component) corresponds to the decrypted phase.

- **Test Vector / Polynomial:** To evaluate a function during bootstrapping, a *test vector* (in FHEW) or *test polynomial* (in TFHE) is constructed. This polynomial's coefficients encode the lookup table of the desired function. During blind rotation, the test polynomial is *rotated* by the encrypted phase, causing the desired function output to appear in the constant term. Duan et al. [2] (Section 1) provides an excellent explanation of this *test-coefficient look-up property*.

- **SampleExtract:** After blind rotation, the result is typically a Ring-LWE (RLWE) or Generalized-LWE (GLWE) ciphertext. The SampleExtract operation then extracts the constant term of the resulting polynomial, converting it back into a standard LWE ciphertext [1] (Algorithm 2). This effectively isolates the function's output.

- **Speed Factors:** TFHE's fast bootstrapping is attributed to:

    - **NTT/FFT Acceleration:** Polynomial multiplications in the underlying ring operations are highly optimized using NTT or FFT, which are efficient for power-of-two polynomial dimensions.

    - **Small Parameters:** Bootstrapping individual gates or small functions allows for smaller cryptographic parameters compared to schemes that bootstrap entire large circuits.

    - **Specialized Operations:** The CMux gate and external product are highly tuned for efficiency.

2.2.5 Programmable Bootstrapping (PBS) / Functional Bootstrapping

A powerful extension of TFHE's bootstrapping is Programmable Bootstrapping (PBS), also known as functional bootstrapping [14, 15]. PBS generalizes the concept by allowing the evaluation of arbitrary functions (represented as lookup tables) while simultaneously refreshing the noise.

Instead of merely refreshing a bit, PBS enables the computation of $f(m)$ from Encrypt(PK, m), where f can be any function. This is achieved by carefully crafting the test polynomial to encode the lookup table of f. This capability is crucial for complex applications beyond simple bitwise operations, such as privacy-preserving machine learning inference where activation functions need to be evaluated. Wang et al. [1] extensively discuss PBS, and Wang et al. [5] mention TFHE-style bootstrapping for comparisons as lookup tables.

2.3 Key Management and Parameterization in TFHE

Effective key management and judicious parameter selection are paramount for balancing the security, correctness, and performance of any FHE scheme, especially TFHE.

2.3.1 Key Types

TFHE schemes utilize several types of keys, each serving a specific purpose in the encryption, evaluation, and noise management processes:

- **Secret Key (SK):** The fundamental key used for decrypting ciphertexts. It is kept private by the data owner.

- **Public Key (PK):** Derived from the SK, this key is publicly available and used by anyone to encrypt data.

- **Evaluation Keys (EK) / Relinearization Keys:** These keys are essential for homomorphic multiplications (e.g., the *External Product* operation). They help manage the size of ciphertexts after multiplication, preventing them from growing too large. Wang et al. [5] mentions relinearization keys rks $\rightarrow s'$ for scheme switching.

- **Bootstrapping Keys (BSK):** These are collections of TGSW ciphertexts that encrypt components of the secret key. They are central to the blind rotation procedure during bootstrapping. The size of BSKs can be substantial, impacting memory footprint and bandwidth [1, 5].

- **Key-Switching Keys (KSK):** Used to transform a ciphertext encrypted under one secret key into a ciphertext encrypting the same plaintext under a different secret key. This is often used to reduce the LWE dimension of a ciphertext or to switch between different secret key structures. Wang et al. [1] (Algorithm 3) provides a detailed description.

- **Rotation Keys:** In SIMD (Single-Instruction, Multiple-Data) operations, multiple plaintext slots are packed into a single ciphertext.

Rotation keys enable permutations or cyclic shifts of these slots, which are necessary for many vectorized algorithms (e.g., convolutions and aggregations). Wang et al. [5] (Section III) discusses their use for ciphertext permutations.

2.3.2 Parameter Selection

The choice of cryptographic parameters directly impacts the security level, the probability of correct decryption, and the computational performance. This is a complex optimization problem, often involving trade-offs:

- **Polynomial Dimension (N):** For RLWE-based schemes like TFHE, operations are performed over polynomial rings $\mathbb{Z}[X]/(X^N+1)$. N is typically a power of two, enabling efficient NTT/FFT-based polynomial multiplication. A larger N generally allows for more complex computations but increases ciphertext size and computational cost. Wang et al. [1] highlights the "rigid coupling" of N with the internal modulus q in traditional TFHE.

- **LWE Dimension (n):** This refers to the dimension of the LWE secret key. A larger n generally increases security but also ciphertext size and computation time.

- **Moduli (q, Q):**
  - **Internal Modulus (q):** The modulus of the input LWE ciphertexts to be bootstrapped. It directly determines the supported message precision (e.g., $q = 2^k$ for k-bit messages). Wang et al. [1] (Section 1) emphasizes how q is traditionally coupled to N.
  - **External Modulus (Q):** The modulus of the GLWE ciphertexts that encrypt the accumulator during bootstrapping. A larger Q provides more room for noise growth but increases the computational cost of operations. Wang et al. [5] (Section 1) notes that Q can grow quadratically with message space in traditional TFHE.

- **Noise Standard Deviation (σ):** The standard deviation of the Gaussian distribution from which noise is sampled during encryption. A smaller σ leads to less noise but can reduce security, while a larger σ increases noise growth. Jin et al. [4] (Section 5.1.5) specifies a Gaussian noise standard deviation of 2^{-15} for their TFHE implementation.

- **Gadget Base (B) and Length (ℓ):** These parameters are used in gadget decomposition. A larger base B or length ℓ can reduce the number of ciphertext operations but might increase noise or key sizes. Wang et al. [1] (Section 5.1) refers to these as B_{pbsk} and ℓ_{pbsk}.

- **Security Level (λ):** This quantifies the computational effort required for an adversary to break the scheme (e.g., 128-bit security means 2^{128} operations). Parameter sets are chosen to meet standard security levels (e.g., 128-bit, 192-bit, 256-bit). Jin et al. [4] (Table 1) shows key size vs. approximate security level.

- **Decryption Failure Rate (DFR):** A critical metric indicating the probability that a ciphertext, after homomorphic operations, cannot be correctly decrypted. Parameters are tuned to ensure a negligible DFR (e.g., 2^{-40} or less). Wang et al. [1] (Section 5.1) provides a detailed analysis of DFR.

The selection of these parameters is often guided by tools like the LWE Estimator [18], which helps translate parameter choices into estimated security levels. The inherent trade-offs mean that optimizing for one aspect (e.g., speed) might compromise another (e.g., key size or DFR). This complex interplay is a central theme in the papers reviewed, as researchers strive to find optimal parameter sets for various applications.

3 Innovations in TFHE Bootstrapping and Algebraic Foundations

The efficiency of Fully Homomorphic Encryption (FHE) schemes, particularly those based on the FHEW/TFHE lineage, is critically dependent on their bootstrapping mechanism. Bootstrapping, the process of refreshing noise in ciphertexts, has been the primary target for optimization since its inception. This section delves into recent, groundbreaking innovations that significantly enhance TFHE bootstrapping by proposing novel cryptosystem configurations, introducing new algebraic foundations, and scaling functional bootstrapping for larger plaintext spaces. These advancements collectively address long-standing challenges related to performance, scalability, key size, and message precision.

3.1 Enhancing Bootstrapping Efficiency through Novel Cryptosystem Configurations

The efficiency of TFHE-like bootstrapping is often limited by the computational cost of underlying homomorphic operations, especially the external product between GSW and LWE ciphertexts. Wang et al. [1], "Accelerating FHEW-like Bootstrapping via New Configurations of the Underlying Cryptosystems" (2025), introduces a novel configuration that significantly accelerates these fundamental building blocks.

3.1.1 Core Contribution: Squared-Gadget GSW with Batching and Scale-Based Operations

Wang et al. [1] propose a new configuration of the GSW Fully Homomorphic Encryption scheme, specifically a "squared-gadget" version, integrated with message batching and scale-based homomorphic operations. This configuration is designed to offer improved efficiency compared to existing approaches by optimizing the structure of the GSW ciphertexts and the method of performing homomorphic multiplication. The core idea is to move away from traditional "fat" gadget matrices to a more compact, squared-matrix form, coupled with a scale-based external product.

3.1.2 Technical Mechanism: Mat-MGSW and Scale-Based External Product

Traditionally, GSW ciphertexts, particularly in their module (MGSW) or ring (RGSW) variants, employ a "fat" gadget matrix G (e.g., a $2 \times \ell$ matrix where $\ell = \log_B Q$) for decomposition. This structure dictates the complexity of operations like the external product.

- **Traditional RGSW/MGSW Structure (Conceptual):** A standard RGSW ciphertext C encrypting a message μ over a polynomial ring R_Q might be represented as:

$$C = (\boldsymbol{a}^\top \quad \boldsymbol{z}\boldsymbol{a}^\top + \boldsymbol{e}^\top) + \mu \cdot G$$

  where G is the gadget matrix. The external product $C \boxdot \boldsymbol{c}$ (between a GSW and an LWE ciphertext $\boldsymbol{c}$) typically involves decomposing $\boldsymbol{c}$ using G^{-1} and then performing matrix-vector multiplication.

- **Proposed Squared-Gadget Mat-MGSW:** Wang et al. [1] introduces a variant where the gadget matrix G is replaced by a scaled identity matrix I_{k+1} (where $k+1$ is the rank of the underlying MLWE scheme), effectively creating a "squared-gadget" structure. The Mat-MGSW ciphertext C encrypting a message matrix M takes the form:

$$C := \left\lfloor B + \frac{M \cdot T}{Q} \cdot I_{k+1} \right\rceil \pmod{T}$$

  where B is a matrix satisfying $(1, -\boldsymbol{s}^\top) \cdot B \approx \boldsymbol{e}$ (for secret $\boldsymbol{s}$ and small noise $\boldsymbol{e}$), T is a larger modulus for MGSW, and Q is the modulus for the MLWE ciphertext. The term T/Q acts as a scaling factor. This makes the gadget matrix effectively $T/Q \cdot I_{k+1}$, a square matrix.

- **Scale-Based External Product:** The external product between this squared-gadget Mat-MGSW ciphertext C and a Vec-MLWE ciphertext $\boldsymbol{c}$ (encrypting m') is defined as:

$$C \boxdot \boldsymbol{c} := \left\lfloor C \cdot \boldsymbol{c} \cdot \frac{Q}{T} \right\rceil \pmod{Q}$$

  This operation is performed by multiplying C and $\boldsymbol{c}$ modulo T, then scaling the result by Q/T and rounding. This "scale-based" approach contrasts with the traditional gadget decomposition-based external product.

Figure 1 demonstrates a visual comparison of a "fat" traditional GSW gadget matrix versus the proposed squared-gadget identity matrix, highlighting the structural difference that underpins the efficiency gains.

Conceptual comparison of traditional "fat" GSW gadget matrix versus the proposed squared-gadget identity matrix. The proposed method replaces a rectangular gadget matrix with a scaled identity matrix, simplifying operations.

3.1.3 Computational Advantages

The shift to a squared-gadget and scale-based external product yields significant computational benefits [1]:

- **Reduced NTT/FFT Transformations:**
  - **Traditional MGSW:** Requires converting the LWE ciphertext to coefficient form (NTT/FFT inverses), performing G^{-1}, converting back to NTT/FFT form, and then performing matrix multiplications. This typically involves $(k+1)$ NTT/FFT inverses and $\ell(k+1)$ NTT/FFT transformations.

- **Proposed Mat-MGSW:** The scale-based external product $C \cdot \boldsymbol{c} \cdot Q/T$ bypasses the complex G^{-1} decomposition. If C and $\boldsymbol{c}$ are already in NTT/FFT form, it primarily involves component-wise multiplications. Conversion to coefficient form for scaling/rounding, followed by conversion back to NTT/FFT form, requires $2(k+r)$ NTT/inv-NTT operations (where r is for batching).

- **Fewer Component-wise Multiplications:** The traditional method involves $(k+1)^2\ell$ component-wise multiplications, while the proposed method needs $(k+r)^2$ (for Mat-MGSW with batching).

- **Overall Cost Reduction:** As shown in Wang et al. [1], Table 3, the number of basic NTT/inv-NTT operations is significantly reduced. For example, for STD128 parameters, traditional OpenFHE might require 10240 NTTs, while their method needs 4096 NTTs (for $r = 1, k = 3, N = 512$).

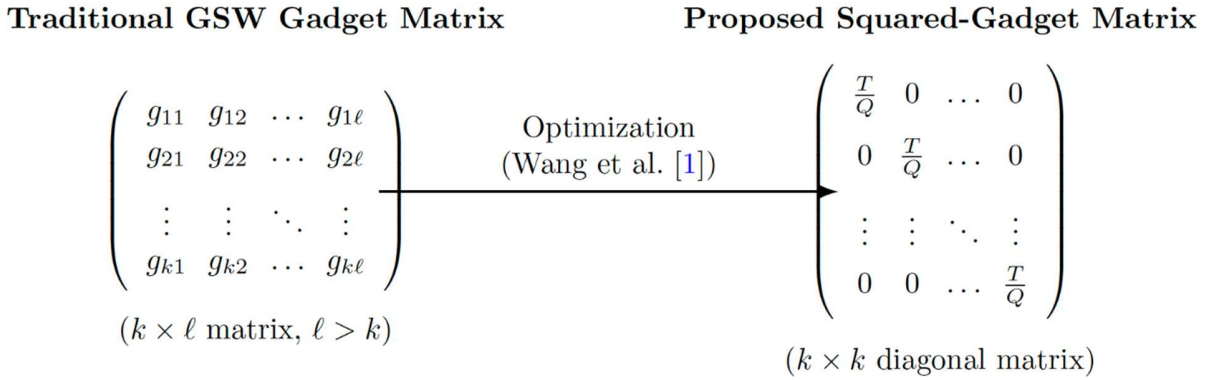

Fig. 1: Conceptual comparison of traditional "fat" GSW gadget matrix versus the proposed squared-gadget identity matrix. The proposed method replaces a rectangular gadget matrix with a scaled identity matrix, simplifying operations.

Performance Metrics and Compatibility

The paper demonstrates substantial concrete performance improvements [1]:

- **Speedup:** More than 2 × faster than current implementations (see Figure 2).
  - **OpenFHE:** Ring-GSW bootstrapping latency under OpenFHE reduced from 84 ms to 26.2 ms.
  - **TFHE:** TFHE bootstrapping latency reduced from 11.4 ms to 4.8 ms.
- **Compatibility:** A key advantage is that the external product implemented by this squared-gadget variant can be directly integrated into existing schemes such as FHEW and TFHE. This enables seamless adoption within current FHE libraries without requiring a complete architectural redesign.
- **Noise Analysis:** The paper provides a detailed noise growth analysis for the scale-based external product, demonstrating that it preserves the asymmetric noise behavior required by FHEW-like systems to ensure polynomially bounded noise growth.

Comparison of bootstrapping execution times for traditional and proposed methods in OpenFHE and TFHE, illustrating the significant speedup achieved by Wang et al. [1].

Critical Observation: While the achieved speedups are substantial, they primarily represent constant-factor improvements. The underlying asymptotic complexity of bootstrapping remains unchanged, which continues to pose challenges for very large-scale or real-time applications. This suggests that further algorithmic breakthroughs or dedicated hardware acceleration will be necessary to move beyond these gains.

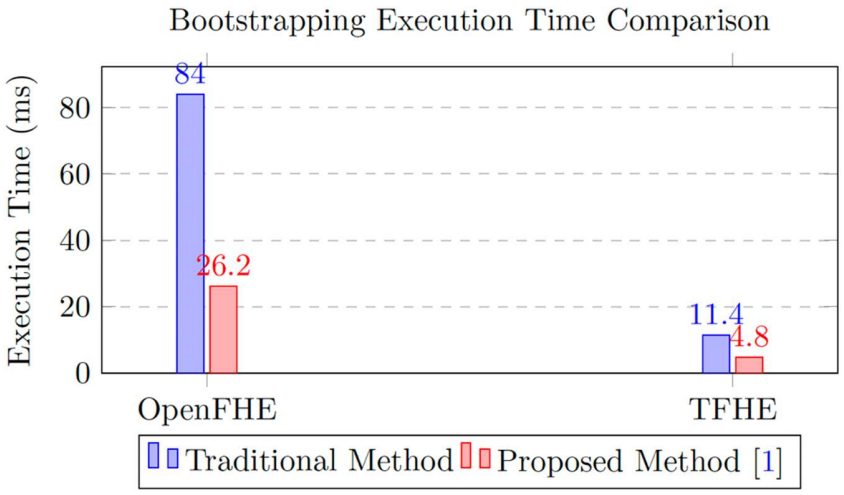

Fig. 2: Comparison of bootstrapping execution times for traditional and proposed methods in OpenFHE and TFHE, illustrating the significant speedup achieved by Wang et al. [1].

3.2 Decoupling Parameters: The Free R-Module Framework for Scalable Bootstrapping

Wang et al. [5], *"Bootstrapping over Free R-Module"* (2025), address a fundamental structural rigidity in FHEW/TFHE: the tight coupling between the polynomial dimension N and the LWE modulus q. This coupling severely limits scalability and performance when targeting higher message precision. To overcome this limitation, the authors introduce a novel algebraic framework that decouples these parameters, enabling more flexible and efficient bootstrapping constructions.

3.2.1 Problem Statement: Rigid Coupling and its Consequences

In traditional FHEW/TFHE bootstrapping, the accumulator structure is the cyclotomic ring

$$R_N = \mathbb{Z}[X]/(X^N + 1).$$

The construction relies on the isomorphism

$$(\mathbb{Z}_q, +) \cong (H, \times),$$

where

$$H = \langle X \rangle = \{X^a \mid a \in [q]\} \subseteq R_{q/2}^{\times}.$$

This mapping requires the LWE ciphertext modulus q to satisfy $q \leq 2N$, which is typically enforced by setting $N = q/2$. This rigid coupling introduces several detrimental consequences [5]:

- **Ring Dimension Expansion:** As message precision increases, the modulus q must increase accordingly, forcing N to grow linearly with q. This inflates the polynomial degree of GLWE ciphertexts, leading to:
    - Larger ciphertext sizes.
    - Increased FFT/NTT sizes, exacerbating computational bottlenecks (e.g., deeper butterfly stages, higher register pressure, and inefficient memory access patterns).
    - Higher computational complexity for ring-based operations.
- **External Modulus Blow-Up:** The noise variance during bootstrapping grows as

  $$O(nkN\sigma^2),$$

  where n is the LWE dimension, k is the GLWE dimension, and σ^2 is the noise variance. To maintain a negligible decryption failure probability, the external modulus Q of GLWE ciphertexts must increase substantially. When $N = q/2$, the modulus grows approximately as

  $$Q = O\big((\sigma q \ln q)^{O(1)}\big),$$

  exhibiting near-quadratic growth with respect to the message space t.

Manuscript received April 5, 2026
Manuscript revised April 20, 2026
https://doi.org/10.22937/IJCSNS.2026.26.4.1

- **Performance and Precision Bottleneck:** The rapid growth of both N and Q inflates computational complexity and key sizes. As a result, practical implementations are constrained to limited bootstrapping precision, such as the ~11-bit precision ceiling observed in the TFHE-rs sparse secret key variant.

3.2.2 Core Contribution: The Free R_N-Module $\oplus_{i=0}^{\tau-1} R_N \cdot X^i$

Wang et al. [5] overcome this limitation by introducing a new accumulator structure: a (finite) free R_N-module of the form

$$\oplus_{i=0}^{\tau-1} R_N \cdot X^i.$$

This generalization decouples N from q through an adjustable parameter τ. For any chosen N, τ is selected such that $q = 2\tau N$. The classical ring construction is recovered when $\tau = 1$.

3.2.3 Technical Mechanism: Module Isomorphism and R_N-Algebra

The paper establishes a sophisticated algebraic framework to enable efficient computation within this new module [5].

- **Module Isomorphism:** The key insight is that the ring $R_{q/2} = \mathbb{Z}[X]/(X^{q/2}+1)$ can be viewed as a free R_N-module. Let $N = q/(2\tau)$. Any polynomial $F(X) \in R_{q/2}$ can be uniquely decomposed as:

$$\begin{aligned} F(X) &= \sum_{k=0}^{q/2-1} a_k X^k \\ &= \sum_{i=0}^{\tau-1} \left(\sum_{j=0}^{N-1} a_{j\tau+i} (X^\tau)^j \right) X^i \\ &= \sum_{i=0}^{\tau-1} F_i(X^\tau) X^i. \end{aligned}$$

By setting $Y = X^\tau$, we have $Y^N = X^{N\tau} = X^{q/2} \equiv -1 \pmod{X^{q/2}+1}$, so $F_i(Y) \in R_N = \mathbb{Z}[Y]/(Y^N+1)$. This establishes the module isomorphism:

$$(\oplus_{i=0}^{\tau-1} R_N \cdot X^i, +) \cong (R_{q/2}, +).$$

This isomorphism $g: \oplus_{i=0}^{\tau-1} R_N \cdot X^i \to R_{q/2}$ allows defining a multiplication $\star$ on the module: for $a, b \in \oplus_{i=0}^{\tau-1} R_N \cdot X^i$,

$$a \star b = g^{-1}(g(a)g(b)).$$

This confers a ring structure on the module:

$$(\oplus_{i=0}^{\tau-1} R_N \cdot X^i, +, \star) \cong (R_{q/2}, +, \times).$$

- **R_N-Algebra and Kronecker Products (Theorem 1):** The module $(\oplus_{i=0}^{\tau-1} R_N \cdot X^i, +, \star)$ is also an R_N-algebra. Theorem 1 in Wang et al. [5] demonstrates how multiplication in a free R-module that is also an R-algebra can be reduced to Kronecker products over the base ring R. For two elements $a = \sum_i a_i \cdot x_i$ and $b = \sum_j b_j \cdot x_j$ in the module (where $a_i, b_j \in R_N$ and x_i are basis elements), their product $c = a \star b = \sum_k c_k \cdot x_k$ has coefficients $(c_0, \dots, c_{\tau-1})$ given by:

$$\begin{aligned} &(c_0, \dots, c_{\tau-1}) = \\ &(a_0, \dots, a_{\tau-1}) \otimes (b_0, \dots, b_{\tau-1}) \cdot M \end{aligned}$$

where $\otimes$ denotes the Kronecker product over R_N, and M is a fixed matrix encoding the multiplication rules for the basis elements $X^i \star X^j = \sum_\ell m_{\ell,i,j} X^\ell$. This means all multiplications in the module can be reduced to multiplications and additions in R_N.

- **Sparsity and Degenerate Multiplications:** While the general formula suggests $O(\tau^3)$ base-ring operations, two crucial optimizations make it efficient [5]:

  a. **Sparsity of M:** The multiplication matrix M is sparse. For basis elements X^i, X^j, their product

  $$X^i \star X^j = Y^{\lfloor (i+j)/\tau \rfloor} \cdot X^k, \quad k = (i+j) \bmod \tau.$$

  This sparsity reduces the cost from τ^3 to τ^2 base ring operations.

  b. **Degenerate Multiplications:** In TFHE bootstrapping, multiplications are often between a monomial plaintext (e.g., X^r) and an accumulator ciphertext, or between an encrypted key and the accumulator. In the latter case, the bootstrapping key encrypts a constant (e.g., s_i), whose module representation has only one non-zero coordinate. This

allows for a "vectorized encryption" optimization where redundant coordinates are replaced by zero blocks. This reduces the computational cost from τ^2 to just τ base multiplications and significantly reduces noise accumulation.

3.2.4 Vectorized Encryption and Homomorphic Operations

The paper defines vectorized encryptions and homomorphic operations over this free R_N-module [5].

- **ACC Vectorized Encryption:**
  - GLWEvec(M) : Encrypts each component $M_i(Y) \in R_N$ of the module representation $(M_0(Y), \dots, M_{\tau-1}(Y))$ individually into a base GLWE ciphertext:

$$\text{GLWEvec}(M) = (\text{GLWE}(M_0), \dots, \text{GLWE}(M_{\tau-1})).$$

  - GGSWvec(M): Similarly for GGSW ciphertexts:

$$\text{GGSWvec}(M) = (\text{GGSW}(M_0), \dots, \text{GGSW}(M_{\tau-1})).$$

- **Vectorized Homomorphic Operations:**
  - **Plaintext-Ciphertext Multiplication:** For a plaintext $M \in \bigoplus_{i=0}^{\tau-1} R_N \cdot X^i$ and a ciphertext $C' = \text{GLWEvec}(M')$, their product $M \cdot C'$ involves a weighted sum of component-wise plaintext-ciphertext multiplications over R_N.
  - **Vectorized External Product:** For $C = \text{GGSWvec}(M)$ and $C' = \text{GLWEvec}(M')$, the vectorized external product $\boxdot_{\text{vec}}$ is defined as:

$$C \boxdot_{\text{vec}} C' = \sum_{i,j} \left(C_i \boxdot C'_j\right) \text{ (over } R_N).$$

  - **Optimization for Bootstrapping:** Due to the degenerate nature of bootstrapping keys (encrypting constants such as s_i), the vectorized encryption simplifies to:

$$\text{GGSWvec}(s_i) = (\text{GGSW}(s_i), \mathbf{0}, \dots, \mathbf{0}).$$

Thus the vectorized external product becomes:

$$C \boxdot_{\text{vec}} C' = (\text{GGSW}(s_i) \boxdot C'_k)_k,$$

reducing the cost to τ base external products and reducing noise accumulation to a single layer.

3.2.5 Asymptotic and Concrete Improvements

The free R_N -module framework yields significant asymptotic and concrete improvements, summarized in Wang et al. [5], Table 1 and Table 3.

- **Decoupling:** The key is that N can now be chosen independently of q. For example, N can be fixed to a hardware-friendly size (e.g., $N = 2048$), and q can be scaled by adjusting τ.
- **External Modulus (Q):**
  - **Traditional TFHE:**

$$Q_{\text{TFHE}} = O\left(\left(\sigma\sqrt{nq\ln q}\right)^{\frac{\ell}{\ell-1}}\right).$$

  - **Proposed Method:**

$$Q_{\text{Ours}} = O\left(\left(\sigma\sqrt{n^2\ln q}\right)^{\frac{\ell}{\ell-1}}\right).$$

  - **Asymptotic Gain:** Reduces Q growth by a factor

$$\left(\frac{\sqrt{q}}{\sqrt{n}}\right)^{\frac{\ell}{\ell-1}}.$$

- **Total Complexity:**
  - **Traditional TFHE:**

$$O(nq\log q).$$

  - **Proposed Method:**

$$O(nq\log N),$$

with $\tau = O(q/N)$ and N chosen as $O(n)$.

  - **Asymptotic Gain:** Reduces complexity by a factor of

$$\frac{\log q}{\log N}.$$

- **Bootstrapping Key Size (BSK):**
  - **Traditional TFHE:**

    $O(nq\log(nq\ln q))$.
  - **Proposed Method:**

    $O(nN\log(nN\ln q))$,

    with $N = O(n)$.
  - **Asymptotic Gain:** Reduces BSK size by a factor

    $$\frac{q\log q}{N\log N}.$$
- **Message Precision and Parallelism:**
  - **Larger Precision:** The reduced Q growth allows for significantly higher message precision. The paper claims increasing the bootstrappable message precision from the 11-bit ceiling to beyond 32 bits under a practical 64-bit Q [5].
  - **Parallelism:** The decomposition into τ sub-ciphertexts enables natural τ-way parallelism, giving a parallel complexity of

    $O(nN\log N)$.
- **Concrete Performance:** The construction achieves remarkable concrete performance [5]:
  - At 11-bit precision, it achieves a **10.71 × speedup** over state-of-the-art TFHE variants (BCLO+).
  - It provides a **707.5 × reduction** in bootstrapping key size at 11-bit precision.
  - It also maintains a **lower decryption failure rate** (2^{-40} vs. 2^{-20}).

3.2.6 Broader Implications

The free R_N-module framework has broader implications for FHE [5]:

- **Automorphism-Based Bootstrapping:** Enables efficient decomposition of automorphisms on $R_{q/2}$ into automorphisms on the smaller ring R_N (Theorem 3 in Wang et al. [5]).
- **PBSManyLUT Integration:** Compatible with PBSManyLUT, offering internal optimizations by discarding redundant sub-accumulators.
- **NTRU Bootstrapping:** Helps keep Q below the "fatigue point" for NTRU-based FHE, potentially enabling higher precision for NTRU.
- **Hardware Acceleration:**

  The fixed polynomial degree, compact and reusable BSK, and intra-ciphertext parallelism make the construction highly hardware-friendly.

Conceptual illustration of hardware utilization for traditional TFHE (where polynomial dimension N varies with message precision) versus the proposed Free R-Module framework (where N can be fixed), showing more consistent and higher utilization for the latter.

Critical Observation: The free R_N-module framework represents a significant advancement in the quest for efficient and scalable FHE. By decoupling the polynomial dimension from the LWE modulus, it offers unprecedented flexibility in parameter selection, enabling higher precision and more efficient use of resources. The substantial reductions in bootstrapping key sizes and the demonstrated speedups in bootstrapping operations are particularly noteworthy.

However, the complexity of the algebraic structures and the need for careful parameter management may pose challenges for implementation and adoption. Additionally, while the framework shows great promise for improving the efficiency of TFHE, the fundamental challenges of FHE, such as the performance overhead of ciphertext arithmetic and the complexity of integrating FHE into existing systems, remain.

Future work should focus on further refining these algebraic techniques, exploring their integration into broader FHE systems, and addressing the remaining challenges to unlock the full potential of FHE in practical, real-world applications.

3.3 Scaling Functional Bootstrapping for Large Plaintexts

While the previous sections focus on fundamental optimizations, Duan et al. [2], "Large-Plaintext

Functional Bootstrapping in FHE with Small Bootstrapping Keys" (2025), tackles the specific challenge of efficiently handling large plaintexts in functional bootstrapping, a crucial capability for complex encrypted computations.

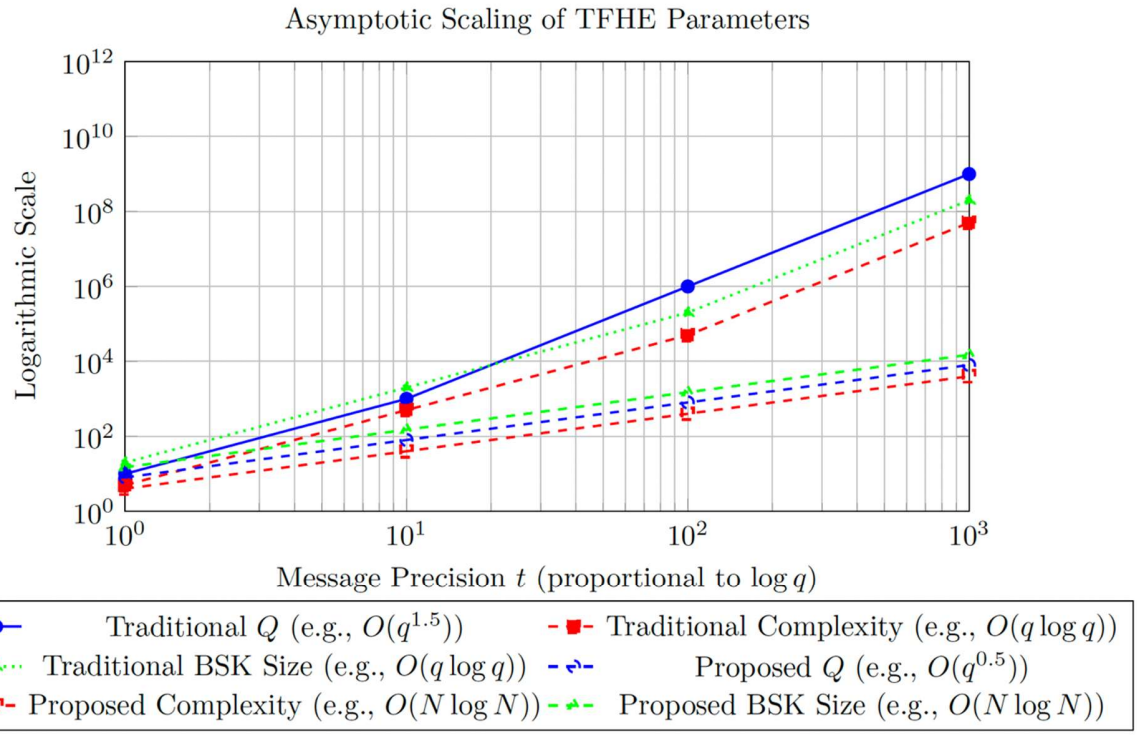

Fig. 3: Conceptual log-log plot illustrating the improved asymptotic scaling of external modulus (Q), total complexity, and Bootstrapping Key (BSK) size for the Free R-Module framework [5] compared to traditional TFHE as a function of message precision.

3.3.1 Problem Statement: Inefficient Large-Plaintext Functional Bootstrapping

In FHEW/TFHE, functional bootstrapping typically encodes the lookup table of a function into the coefficients of a single test polynomial. For large plaintexts (i.e., when the range of inputs q is large), this requires a very high-degree polynomial [2].

- **High Polynomial Degree:** A large q directly translates to a large polynomial degree N for the test polynomial.
- **Increased Bootstrapping Time:** Higher N leads to slower computation due to larger FFT/NTT operations.
- **High Memory Cost:** Bootstrapping keys (BSK) and key-switching keys (KSK) scale with N, leading to substantial memory consumption.
- **Limited Functionality:** Existing strategies for large plaintexts (e.g., block-wise bootstrapping) are often limited to specific functions such as identity or sign.

3.3.2 Core Contribution: Monic Monomial Permutation Matrices (MMPM)

Duan et al. [2] proposes a novel FHEW/TFHE framework for large-plaintext functional bootstrapping by encoding the lookup table of any function in a polynomial vector, whose coefficients can hold more data. This

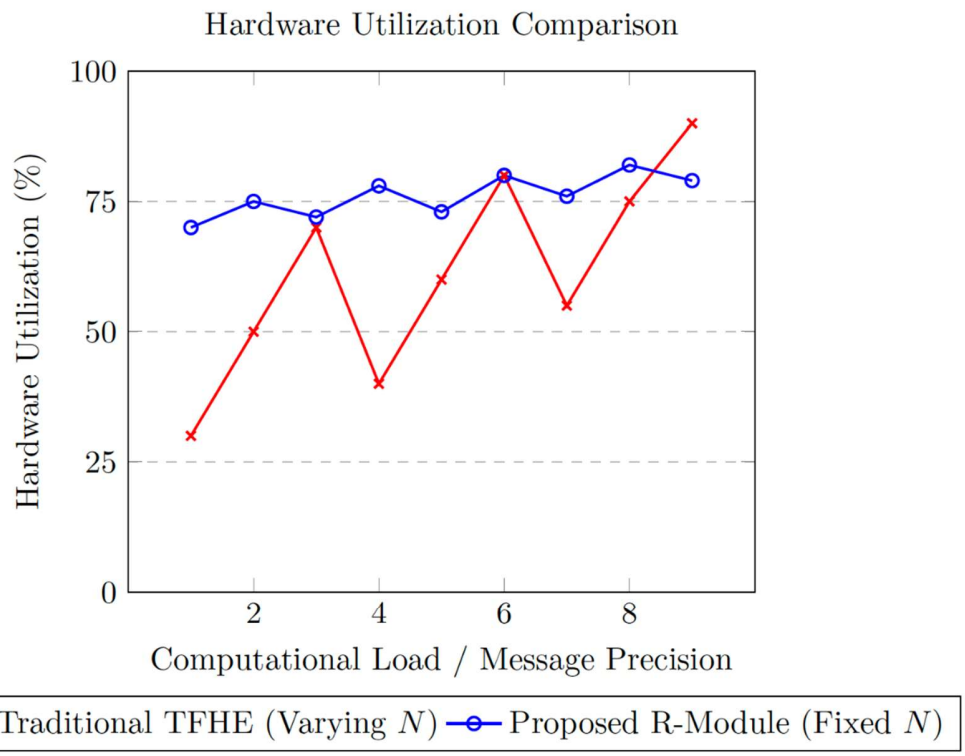

Fig. 4: Conceptual illustration of hardware utilization for traditional TFHE (where polynomial dimension N varies with message precision) versus the proposed Free R-Module framework [5] (where N can be fixed), showing more consistent and higher utilization for the latter.

is achieved by representing the additive group $\mathbb{Z}_q$ using *Monic Monomial Permutation Matrices* (MMPM).

3.3.3 Technical Mechanism: Polynomial Vectors and MMPM Representation

- **Polynomial Vector Encoding:** Instead of a single polynomial, the lookup table for a function f (extended to f' on $\mathbb{Z}_{q'}$) is encoded into an r-dimensional polynomial vector [2]. If N is the maximal efficient ring dimension and $q' = 2Nr \geq q$, the vector has the form:

$$\text{testf} = \begin{pmatrix} f'(0) + f'(1)x + \cdots + f'(N-1)x^{N-1} \\ f'(N) + f'(N+1)x + \cdots + f'(2N-1)x^{N-1} \\ \vdots \\ f'((r-1)N) + \cdots + f'(rN-1)x^{N-1} \end{pmatrix}$$

where each entry is a polynomial in $R_N = \mathbb{Z}[x]/(x^N + 1)$.

- **MMPM Representation of $\mathbb{Z}_q$:** The paper introduces Monic Monomial Permutation Matrices (MMPMs) as a new representation space for $\mathbb{Z}_q$ [2]. An $r \times r$ MMPM is a matrix where each row and column has exactly one non-zero entry, and each such entry is a monic monomial (e.g., x^u). The representation $\Phi: \mathbb{Z}_{q'} \to \text{MMPM}_r$ is constructed so that $\Phi(1)$ is similar to a block-diagonal matrix whose blocks are cyclic-shift matrices with monic monomials. A key example is:

$$\Phi(1) = \begin{pmatrix} 0 & I_{(r-1)\times(r-1)} \\ x & 0 \end{pmatrix}$$

The order of this matrix is $r \cdot \text{order}(x) = 2Nr = q'$. This integrates the permutation-matrix representation of AP14 (when $x = 1$) and the monic-monomial representation of FHEW/TFHE (when $r = 1$).

- **Test-Coefficient Look-up Property:** The action of $\Phi(c)$ (for $c = \Delta m + e$) on the testf

Manuscript received April 5, 2026
Manuscript revised April 20, 2026
https://doi.org/10.22937/IJCSNS.2026.26.4.1

polynomial vector, via matrix–vector multiplication, has the key property that the *first entry* of the resulting vector contains $f(m)$ as its constant term. This property is fundamental to functional bootstrapping [2].

Illustration of a polynomial vector being acted upon by a Monic Monomial Permutation Matrix (MMPM). The multiplication shifts the desired function output to the constant term.

3.3.4 TecKey Size and Runtime Improvements

The MMPM framework offers significant efficiency gains [2]:

- **Bootstrapping Key Size:** For $r = \text{poly}(n)$ and $N = \text{poly}(n)$, the total number of bits in the bootstrapping keys is

$$\frac{8(n\ell BN\log Q)}{r}.$$

This is a polynomial factor smaller than TFHE's $8n\ell BN\log Q$.

- **Key-Switching Key Size:** Similarly, the key-switching key size is

$$\frac{NB_{KS}\ell_{KS}(n+1)\log Q}{r},$$

also a polynomial factor smaller.

- **Phase Accumulation Time Complexity:** The phase accumulation procedure (blind rotation) involves n CMux operations. Each CMux now requires $2r$ multiplications between an RLWE ciphertext and an RGSW ciphertext. The overall integer multiplication complexity is

$$12n\ell BN\log(N/r),$$

which is a constant-factor improvement over TFHE's $12n\ell BN\log N$ (with N replaced by N/r for fair comparison).

Configuration	BGV Amortized Time (s)	TFHE Estimated Time (s)
(1,1)	0.1	1.0
(2,2)	0.5	5.0
(3,3)	5.0	50.0
(4,4)	66.44	76.05
(5,5)	10.0	100.0

Amortized execution times for various Banded Smith-Waterman (BSW) configurations using BGV (Deogade) versus TFHE estimates (Bataa et al. [19]). BGV shows superior performance, especially in lower-dimensional cases.

3.3.5 Experimental Results and TFHE Comparison

Deogade's work provides a direct comparison against a TFHE-based Smith-Waterman implementation by Bataa et al. [19] (2020) and estimates for various BSW configurations [3].

- **BGV Performance for BSW:**
  - For (4,4) [3] BSW (Smith-Waterman for sequences of length 4, band size 3), Deogade's BGV implementation achieves an amortized time of 8.88 seconds with OpenMP parallelism [3].
  - This compares favorably to Bataa et al.'s TFHE implementation [19], which took 64s (sequential circuits) and 32s (parallel circuits for HE operations).
  - This represents a 10× speedup over the TFHE estimate and is only 1776 × slower than unencrypted BSW (compared to TFHE's 6400× slower) [3].
- **Amortized Time Comparison** [3, Table 3,7, 3.8]: The BGV approach consistently yields lower amortized times across various (reference length, query length)[bandsize] configurations, including (1,1) [1], (2,2) [1], (3,3) [1], (3,3) [2], (4,4) [1], (4,4) [3], and (5,5) [1]. For example, (3,3) [2] BSW takes 66.44s (BGV) vs. estimated 76.05s (TFHE).

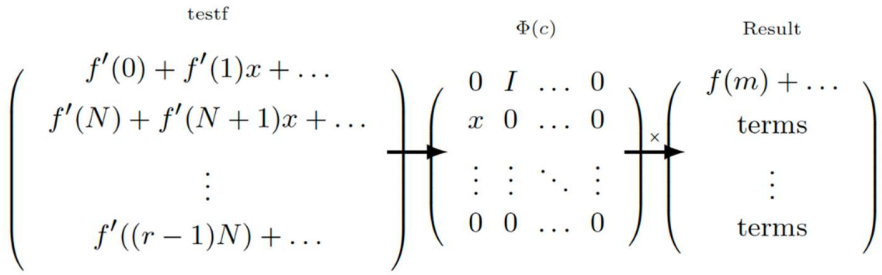

Fig. 5: Illustration of a polynomial vector being acted upon by a Monic Monomial Permutation Matrix (MMPM). The multiplication shifts the desired function output to the constant term.

Configuration	BGV Amortized Time (s)	TFHE Estimated Time (s)
(1,1)	0.1	1.0
(2,2)	0.5	5.0
(3,3)	5.0	50.0
(4,4)	66.44	76.05
(5,5)	10.0	100.0

Fig. 6: Amortized execution times for various Banded Smith-Waterman (BSW) configurations using BGV (Deogade [3]) versus TFHE estimates (Bataa et al. [22]). BGV shows superior performance, especially in lower-dimensional cases.

- **Parameters:** BGV parameters are chosen for ≈ 80 -bit security, with plaintext modulus p selected based on the maximum possible score (e.g., $p = 41$ for (4,4) BSW) [3].
- **Memory Footprint:** For (3,3) [2] BSW, the input message size (6 ciphertexts) is ≈ 2.2 GB, and output message size (3 ciphertexts) is ≈ 1.1 GB [3].

3.3.6 Partial Conversions for Other GenomicsBench Kernels

Deogade's thesis also explores partial conversions of other compute-intensive GenomicsBench kernels—chain and fmi—to the homomorphic encryption domain using the BFV scheme. These experiments highlight the significant challenges of porting more complex, control-heavy, and less arithmetic-friendly algorithms to HE [3].

- **Chain Kernel:**
  - **Purpose:** A one-dimensional dynamic programming algorithm used to find the best parent for overlapping seeds and stitch them into longer sequences [3].
  - **Challenges:** Involves complex bitwise operations (logical AND, right shift), floating-point operations (e.g., avg_qspan), and table lookups (e.g., logarithm tables) [3], none of which are HE-friendly [3].
  - **Partial Solution:** Floating-point values are quantized by scaling with 2^{10} and rounding. Bitwise operations and table lookups are often assumed to be computed by the client and provided as encrypted inputs, or approximated with polynomial interpolation [3].

Manuscript received April 5, 2026
Manuscript revised April 20, 2026
https://doi.org/10.22937/IJCSNS.2026.26.4.1

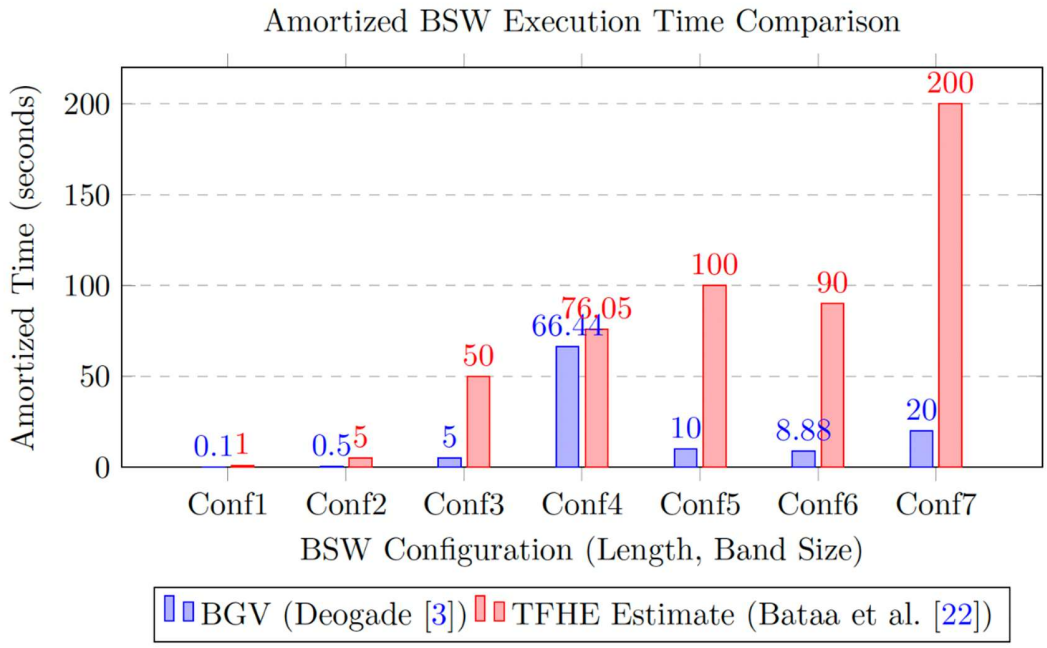

Fig. 7: Amortized execution times for various Banded Smith-Waterman (BSW) configurations using BGV (Deogade [3]) versus TFHE estimates (Bataa et al. [22]). BGV shows superior performance, especially in lower-dimensional cases.

- **Performance:** A single iteration of finding a parent seed requires $O(100)$ seconds under HE, compared to 200–500 ns unencrypted. Processing all seeds in a read requires 49.7 hours in HE, which is $\sim 7.2 \times 10^6$ times slower than the $24801\,\mu$s plaintext runtime [3].

- **FMI Kernel (FM-Index Search):**
  - **Purpose:** Locating substrings within a reference genome using FM-index search [3].
  - **Challenges:** Irregular array indexing, character-by-character processing [3].
  - **Partial Solution:** One ciphertext per character is required (instead of SIMD batching), causing memory usage to balloon from 1 byte to ~1 MB per character. Intermediate steps such as comparisons and bit shifts often require decryption to proceed [3].
  - **Performance:** Processing 5 reads takes 27241.65 seconds under HE, approximately 3700 × slower than the plaintext runtime of 7.32 seconds [3].

Implications:The genomics case studies demonstrate that while SIMD-optimized HE (like BGV) can achieve competitive amortized performance for structured arithmetic tasks like BSW, the overhead for complex, non-arithmetic-friendly operations (bitwise logic, table lookups, irregular memory access) remains extremely high, often necessitating partial decryption or significant architectural compromises [3].

Critical Observation: The substantial slowdowns observed for the Chain and FMI kernels highlight a critical limitation of current FHE schemes: their inefficiency for algorithms that rely heavily on bitwise operations, irregular memory access patterns, or complex control flow. While TFHE excels at bit-level operations and comparisons, its performance for these specific genomic tasks was not directly evaluated in this work, but the general challenges remain. This suggests that a hybrid approach combining FHE with other privacy-enhancing technologies or specialized FHE schemes for bitwise logic might be necessary for broader genomic privacy. The large memory footprint also remains a practical barrier.

Manuscript received April 5, 2026
Manuscript revised April 20, 2026
https://doi.org/10.22937/IJCSNS.2026.26.4.1

3.4 TFHE and FHE in Cloud Computing: A System Perspective

Yadavalli et al. [6], "Homomorphic Encryption Methods Applied to Cloud Computing: A Practical Architecture for Elastic, Verifiable Confidential Compute" (2024), shifts focus from specific cryptographic optimizations to a holistic system perspective, proposing HELIOS, a cloud framework that integrates FHE as a first-class primitive for confidential computing. This paper highlights the crucial role of TFHE-style bootstrapping for specific operations within a broader FHE ecosystem.

3.4.1 Problem Statement: Bridging Cryptographic Feasibility and Cloud Practicality

Despite significant FHE advancements, deploying HE at cloud scale faces several challenges [6]:

- **Performance Gap:** Ciphertext arithmetic is orders of magnitude slower than plaintext compute.
- **Algorithmic Discipline:** Noise growth and rescaling require careful circuit design.
- **Program Layout Complexity:** Vectorization and rotation keys complicate program layout and optimization.
- **Lack of Verifiability:** Bare HE provides confidentiality but not verifiability, meaning the cloud could err or equivocate without detection.
- **Resource Management:** Integrating HE's unique cost structure into elastic, multi-tenant cloud schedulers is complex.

3.4.2 HELIOS: A Cloud-Native FHE Architecture

HELIOS proposes a principled, cloud-native framework for applying homomorphic encryption at scale, emphasizing efficiency, verifiability, and operational ergonomics. It treats FHE as a schedulable, verifiable service primitive [6].

- **Hybrid FHE Scheme Selection:** HELIOS employs a compiler that selects among multiple HE schemes based on the operator requirements [6]:
  - **CKKS:** Used for floating-point style analytics, signal processing, and model inference, where bounded numerical error is acceptable.
  - **BFV/BGV:** Used for exact modular arithmetic, integer computations, and counter-based operations.
  - **TFHE-style Bootstrapping:** Used for comparisons, joins, filters, and argmax. These operations are implemented either via TFHE programmable bootstrapping on fixed-point encodings, or via CKKS polynomial approximations of $\mathrm{sgn}(x)$ followed by low-precision TFHE refinement. This highlights TFHE's specialization for non-linear and comparison-heavy workloads.
- **Compiler and Dataflow Graphs:** HELIOS begins with a compiler that lowers user programs into acyclic dataflow graphs.
  - **CKKS Scaling Management:** For CKKS circuits, the compiler manages scale factors Δ and the modulus chain $Q = \prod_{i=0}^{L} q_i$ to bound numerical error [6]:

$$\|\epsilon_{\mathrm{tot}}\|_2 \le \gamma_0 \Delta^{-1} + \sum_{\ell=1}^{L} \gamma_\ell \prod_{j=1}^{\ell} \kappa_j,$$

    where κ_j are Lipschitz constants and γ_ℓ capture rounding and rescaling errors.
  - **BFV/BGV Noise Management:** For BFV/BGV circuits, the compiler ensures noise remains bounded through modulus switching using:

$$B_{\ell+1} \le \alpha B_\ell + \beta,$$

    where B_ℓ denotes noise magnitude after the ℓ-th operation.
- **Key Management and Rotations:**
  - A dedicated cryptographic key service supports threshold issuance and rotation-key escrow while ensuring it never learns plaintexts [6].
  - Ciphertext permutations (rotations) rely on rotation keys selected through a windowed baby-step/giant-step planning strategy, reducing key material and runtime.

- **Storage Layer:** The system packs encrypted columnar data into ciphertext SIMD lanes, enabling efficient multi-tenant computation and amortizing FHE costs across workloads [6].

3.4.3 Verifiability through HE-MACs and SNARKs

A critical contribution of HELIOS is addressing the lack of verifiability in bare FHE [6].

- **Homomorphic Message Authentication Tags (HE-MACs):** For each ciphertext ct, the client computes a tag $\tau = \mathrm{MAC}_k(\mathrm{Com}(ct))$ over a binding commitment $\mathrm{Com}(ct)$ under a secret key k [6].

- **Tag Updates and Succinct Proofs:** Homomorphic updates carry companion tags, which are updated by the server as affine functions. For non-linear steps (e.g., multiplication and bootstrapping), the server emits a succinct non-interactive proof (SNARK) π attesting that the tag update follows the protocol specification [6].

- **Verification:** The verification equation reduces to bilinear checks over commitments to ring elements, enabling SNARK-friendly constraint systems. The client accepts the output $(ct_{\mathrm{out}}, \tau_{\mathrm{out}}, \pi)$ if and only if π verifies and τ_{out} is consistent with ct_{out} [6].

3.4.4 Scheduling and Cost Model

HELIOS integrates a sophisticated cost model into its scheduler to manage elastic provisioning and meet latency objectives under price constraints [6].

- **Cost Model:** HELIOS employs a linear model to estimate execution time [6]:

  $$T \approx \alpha M_{\times} + \beta M_{+} + \gamma R + \delta \mathrm{Boot} + \eta K,$$

  where $M_{\times}$, M_{+}, R, Boot, and K denote the number of homomorphic multiplications, additions, rotations, bootstrapping operations, and keyswitches, respectively. The coefficients $(\alpha, \beta, \gamma, \delta, \eta)$ are hardware- and parameter-dependent and are learned online.

- **Resource Management:** The control plane selects serverless workers equipped with NTT-accelerated back-ends and GPUs, batches independent NTT operations, and dynamically adjusts batching and concurrency to enforce Service Level Objectives (SLOs) [6].

- **Multi-Party Computation:** HELIOS leverages threshold homomorphic encryption, in which secret keys are distributed using Shamir secret sharing, relinearization keys are additively combined, and decryption requires participation from a quorum of parties [6].

3.4.5 Performance and Security Analysis

HELIOS preserves the cryptographic guarantees of the underlying lattice schemes while providing practical performance for cloud-native FHE workloads [6].

- **Security:** The system's security reduces directly to the hardness of the Ring-Learning-With-Errors (RLWE) problem at the chosen ring dimension N and ciphertext modulus Q. All evaluation keys (relinearization, rotation, bootstrapping keys) maintain *IND-CPA* security through fresh noise injection introduced via key-switching operations, preventing noise-pattern leakage [6].

- **Accuracy for CKKS:** For approximate arithmetic, the end-to-end error is bounded by CKKS encoding, polynomial approximation errors, and the cumulative effect of rescaling. Accuracy depends on the initial scale Δ and the Lipschitz sensitivity of the circuit:

  $$\|\varepsilon_{\mathrm{tot}}\|_2 \;\le\; \gamma_0 \Delta^{-1} \;+\; \sum_{\ell=1}^{L} \gamma_\ell \prod_{j=1}^{\ell} \kappa_j,$$

  where κ_j are layer-wise Lipschitz constants and γ_ℓ account for rounding and rescaling noise [6]. The compiler automatically selects Δ and the modulus chain to keep the final error within user-specified tolerances.

- **Performance:** Core primitive costs follow the asymptotics of the underlying rings:

  - Number-Theoretic Transforms (NTTs) for polynomial multiplication require

    $O(N \log N)$ field operations.

  - TFHE-style bootstrapping dominates deep circuits, with empirical latency

    $$T_{\mathrm{bs}} \approx c_3\, n \log n,$$

    where n is the LWE dimension and c_3 is a hardware- and backend-dependent constant determined by FFT/NTT

implementations. GPU acceleration substantially reduces c_3, yielding large practical speedups.

- **Experimental Indications:** Across realistic encrypted analytics and neural-inference workloads, HELIOS sustains 128-bit RLWE security and achieves:
    - **sub-second P95 latency** for circuits of depth six to eight,
    - with only **one or two TFHE bootstraps** in the critical path,
    - while supporting full verifiability and multi-party key distribution [6].

Implications: HELIOS demonstrates a practical path toward *confidential-by-default* cloud services, wherein encrypted computation becomes the norm and plaintext exposure is eliminated from the cloud trust boundary. Through its hybrid FHE architecture, scheme-specialized compiler, and verifiable execution model, the system shows that large-scale encrypted analytics can remain *efficient*, *elastic*, and *auditable* [6]. A key insight is the strategic positioning of TFHE-style bootstrapping as an indispensable primitive for comparison-heavy and integrity-critical operations—joins, filters, argmax, conditional logic—within a broader orchestration of CKKS and BFV/BGV. This establishes TFHE not as a standalone solution but as a vital component in a heterogeneous, service-oriented FHE stack.

Critical Observation: Despite its promise, deploying and managing an architecture as sophisticated as HELIOS remains challenging. While the system leverages homomorphic message-authentication tags and succinct non-interactive proofs (SNARKs) for verifiable execution, the real-world cost of SNARK generation and verification can be substantial. For deeper or more complex dataflow graphs, these cryptographic proofs may threaten the sub-second latency targets that the system aspires to deliver.

Moreover, compiler and scheduling infrastructures capable of optimizing FHE circuits, managing precision (CKKS), controlling noise (BFV/BGV), and orchestrating bootstrapping (TFHE) are still under active research. The maturity of such tools will heavily influence the practicality of cloud-native encrypted computation.

Finally, integrating the algebraic advances from recent FHE research—including the programmable bootstrapping optimizations from Wang et al. [5], the MMPM representation from Duan et al. [2], and the modular free-R_N constructions from Wang et al. [1]—could further improve TFHE performance within HELIOS. Such integration may reduce the constants in T_{bs}, shrink bootstrapping key sizes, and simplify key management, bringing fully confidential cloud services closer to widespread deployment.

3.5 Synthesis of Practical Challenges and Emerging Solutions

The practical applications discussed in this section, from securing federated learning to enabling confidential genomics and cloud computing, collectively underscore a critical synthesis of challenges and emerging solutions in the FHE landscape, with TFHE playing a pivotal role.

- **Hybrid FHE Strategies are Essential:** The reviewed papers consistently advocate for hybrid approaches. DMAFL [4] combines CKKS-based FHE with PVSS and blockchain, while HELIOS [6] integrates CKKS, BFV/BGV, and TFHE-style bootstrapping. This confirms that no single FHE scheme is universally optimal; rather, combining their strengths (e.g., CKKS for approximate floating-point arithmetic, BFV/BGV for exact integers, and TFHE for fast bootstrapping and comparisons) is key to addressing diverse application requirements. The algebraic innovations in bootstrapping [1, 5] directly enhance the efficiency of the TFHE components within these hybrid systems, making such integrations more viable.

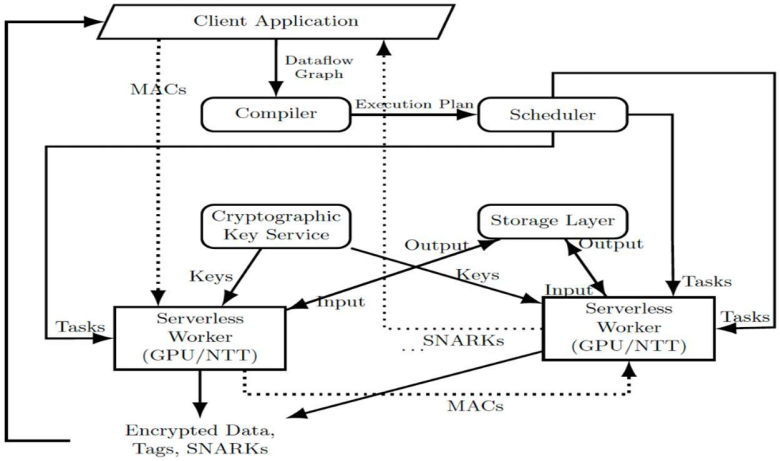

Fig. 8: System architecture diagram of HELIOS [6], a cloud-native framework for elastic, verifiable confidential computing. The framework integrates a compiler, scheduler, cryptographic key service, storage layer, and serverless workers, with homomorphic message authentication codes (MACs) and SNARKs ensuring verifiability.

- **Performance vs. Functionality Trade-offs Remain Significant:** The genomics work [3] starkly illustrates the performance penalties associated with complex, non-arithmetic-friendly operations such as bitwise shifts, table lookups, and irregular memory access. While SIMD-optimized BGV excels at structured arithmetic (e.g., Banded Smith–Waterman), kernels such as chain and fmi exhibit orders-of-magnitude slowdowns, often necessitating partial decryption or client-side computation. This highlights that while FHE is theoretically capable of evaluating arbitrary functions, its practical efficiency is highly dependent on how well the target computation maps to homomorphic primitives. Even with the constant-factor speedups from [1], the fundamental gap for certain classes of operations persists.

- **The Critical Role of Bootstrapping (and its Optimization):** Despite its computational cost, bootstrapping remains indispensable for arbitrary-depth computations and non-linear operations such as comparisons. TFHE's fast, gate-by-gate bootstrapping, along with its refined variants, is consistently chosen for these critical steps in hybrid systems (e.g., HELIOS [6] for comparisons and DMAFL [4] within its CKKS-based FHE pipeline). The advancements in [1, 2, 5] directly target this bottleneck, delivering substantial improvements in speed, key size, and message precision, thereby expanding the feasibility of complex bootstrapped workloads.

- **Beyond Cryptography: System-Level Integration is Paramount:** Effective FHE deployment extends beyond cryptographic primitives to include robust system architectures. Blockchain integration in DMAFL [4] addresses decentralized trust and auditability, while compiler-driven optimization, cost modeling, and elastic scheduling in HELIOS [6] are essential for cloud-scale adoption. These system-level considerations are as critical as cryptographic innovations themselves in achieving practical, deployable FHE systems.

- **Verifiability as a Growing Requirement:** As FHE moves into untrusted execution environments, verifiable computation becomes increasingly important. HELIOS [6] addresses this gap through the integration of homomorphic message authentication codes (HE-MACs) and succinct non-interactive arguments of knowledge (SNARKs), enabling integrity verification of encrypted computation. This represents a significant step toward auditable confidential computing and addresses a fundamental limitation of bare FHE.

- **Parameter Scalability is Being Actively Addressed:** The innovations presented in [1, 5] directly confront the parameter scalability problem by decoupling the polynomial dimension N from the LWE modulus q, enabling higher precision with smaller key sizes and improved performance. These advances are crucial for extending FHE to larger plaintext domains and more complex functions. The key size reductions achieved via MMPM-based bootstrapping [2] further enhance deployability, particularly in memory- and bandwidth-constrained environments.

4. Conclusion

The landscape of Fully Homomorphic Encryption (FHE) has undergone a profound transformation since its inception [7], evolving from a theoretical marvel into a tangible technology with the potential to redefine privacy in the digital age. This review has synthesized six recent and impactful research papers [1, 2, 3, 4, 5, 6], offering a focused look at the cutting-edge advancements in TFHE (Torus FHE) and its broader FHE context. Our exploration has traversed the intricate terrain of algebraic innovation, performance optimization, and application-specific integration, revealing a vibrant research ecosystem dedicated to overcoming the formidable challenges of practical FHE deployment.

A central theme emerging from this review is the relentless pursuit of efficiency and scalability in TFHE bootstrapping. Wang et al. [1] demonstrate significant gains through novel cryptosystem configurations, achieving over $2\times$ speedups by optimizing the underlying GSW structure. More fundamentally, Wang et al. [5] introduce a groundbreaking free R-module framework that directly confronts the long-standing "rigid coupling" between the polynomial dimension and the LWE modulus in traditional TFHE. This innovation not only promises unprecedented precision (beyond 32 bits) but also delivers remarkable concrete performance, showcasing $10.71\times$ speedups and $707.5\times$ smaller key sizes at 11-bit precision. Complementing these efforts, Duan et al. [2] address the scalability of functional bootstrapping for large plaintexts, achieving polynomial reductions in key sizes and constant-factor runtime improvements through the ingenious use of Monic Monomial Permutation Matrices (MMPM). These algebraic and structural advancements collectively underscore a paradigm shift towards more flexible, efficient, and scalable TFHE primitives.

Beyond core cryptographic optimizations, the reviewed literature vividly illustrates the strategic integration of TFHE into high-impact application domains. In federated learning, Jin et al. [4] present DMAFL, a robust framework that leverages a CKKS-based Threshold FHE scheme, blockchain, and multi-level verification to fortify federated learning against model poisoning and privacy leakage. DMAFL's superior accuracy retention under attack and significant reduction in encryption and decryption overhead highlight the power of hybrid cryptographic and architectural designs. Similarly, Deogade [3] showcases the application of FHE (with TFHE as a comparative benchmark) to the sensitive field of genomics, demonstrating how SIMD-optimized BGV can achieve competitive amortized performance for the Banded Smith–Waterman algorithm, while also revealing the persistent performance challenges for less arithmetic-friendly genomic kernels.

Finally, the vision of FHE as a first-class primitive in cloud computing is brought into sharp focus by Yadavalli et al. [6]. Their proposed HELIOS architecture exemplifies a holistic approach, integrating hybrid FHE schemes (including TFHE-style bootstrapping for comparisons), homomorphic Message Authentication Tags (MACs) for verifiability, and a sophisticated cost model for elastic and auditable confidential compute. This work underscores the necessity of moving beyond cryptographic primitives alone to encompass compiler optimizations, resource scheduling, and robust system-level integration.

In summary, the journey of TFHE and its FHEW lineage reflects a dynamic and rapidly evolving field. The innovations presented in these six papers [1, 2, 3, 4, 5, 6] collectively push the boundaries of what is practically achievable with FHE. They demonstrate a clear trend towards:

- **Hybrid Scheme Utilization:** Leveraging the strengths of different FHE schemes (CKKS for floating-point arithmetic, BFV/BGV for exact integers, and TFHE for fast bootstrapping and comparisons) within a single application.
- **Parameter Decoupling and Scalability:** Developing new algebraic structures that break rigid parameter dependencies, enabling higher precision and more efficient resource utilization [1, 5].
- **System-Level Co-design:** Integrating FHE with complementary technologies such as blockchain [4] and verifiable computation [6], and embedding it within robust cloud architectures [6].
- **Amortization and Parallelism:** Maximizing efficiency through SIMD packing, batching, and exploiting inherent parallelism in new module structures [3, 5].

While significant progress has been made, several challenges persist, necessitating continued research:

- **Performance for Complex Control Flow and Bitwise Operations:** Despite advancements, FHE remains inefficient for algorithms with irregular memory access, complex branching logic, or fine-grained bitwise operations, as highlighted in the genomics applications [3]. Hybrid approaches with other privacy-enhancing technologies or dedicated FHE schemes for bitwise logic are still needed.

- **Developer Experience and Tooling:** The complexity of FHE parameter selection, circuit design, and optimization remains a barrier to broader adoption. More user-friendly compilers, debuggers, and high-level programming abstractions are crucial for wider accessibility.

- **Standardization and Interoperability:** As the FHE landscape diversifies with multiple schemes and libraries, standardization efforts are essential to ensure interoperability and reduce fragmentation.

- **Post-Quantum Security Evolution:** While current FHE schemes are lattice-based and considered post-quantum secure, continuous monitoring and adaptation to emerging cryptanalytic advances are necessary.

- **Scalability for Real-Time and Large-Scale Systems:** Even with recent speedups and key reductions, FHE overhead still limits applicability in ultra-low-latency or extremely high-throughput settings. Advances in hardware acceleration, specialized co-processors, and cryptographic primitives remain critical.

The path towards confidential-by-default computing is steadily being paved, and TFHE is unequivocally at its forefront. Addressing these remaining challenges will be essential for FHE to transition from specialized deployments to pervasive adoption across diverse privacy-sensitive domains.

References

[1] Wang, H., Luo, M., Xia, H., Wang, M., & Hou, H. (2025). Accelerating FHEW-like Bootstrapping via New Configurations of the Underlying Cryptosystems.

[2] Duan, K., Li, H., Liu, D., & Ma, G. (2025). Large-Plaintext Functional Bootstrapping in FHE with Small Bootstrapping Keys.

[3] Deogade, K. B. (2025). Efficient System Design With Homomorphic Encryption.

[4] Jin, C., Wang, J., Lu, W., Luo, C., Liu, J., Chen, G., & Zhai, Z. (2025). DMAFL: Effective defense against malicious attacker federated learning framework via blockchain and TFHE. Journal of King Saud University - Computer and Information Sciences, 37, 231.

[5] Wang, R., Bai, J., Liu, Y., Zhang, X., Lu, X., Zhao, L., Wang, K., & Hou, R. (2025). Bootstrapping over Free R-Module.

[6] Yadavalli, R., Solomon, J., & Sharma, V. (2024). Homomorphic Encryption Methods Applied to Cloud Computing: A Practical Architecture for Elastic, Verifiable Confidential Compute.

[7] Gentry, C. (2009). Fully Homomorphic Encryption Using Ideal Lattices. In Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing (pp. 169–178). ACM.

[8] Gentry, C., & Halevi, S. (2011). Implementing Gentry's Fully-Homomorphic Encryption Scheme. In Annual International Conference on the Theory and Applications of Cryptographic Techniques (pp. 129–148). Springer.

[9] Regev, O. (2009). On Lattices, Learning with Errors, Random Linear Codes, and Cryptography. Journal of the ACM (JACM), 56(6), 1–40.

[10] Lyubashevsky, V., Peikert, C., & Regev, O. (2010). On Ideal Lattices and Learning with Errors over Rings. In Annual International Conference on the Theory and Applications of Cryptographic Techniques (pp. 1–23). Springer.

[11] Hoffstein, J., Pipher, J., & Silverman, J. H. (1998). NTRU: A Ring-Based Public Key Cryptosystem. In International Workshop on Ant Colony Optimization and Swarm Intelligence.

[12] Chillotti, I., Gama, N., Georgieva, M., & Izabachène, M. (2016). Faster Fully Homomorphic Encryption: Bootstrapping in Less Than 0.1 Seconds. In Advances in Cryptology – ASIACRYPT 2016 (pp. 3–33). Springer.

[13] Gentry, C., Sahai, A., & Waters, B. (2013). Homomorphic Encryption from Learning with Errors: Conceptually-Simpler, Asymptotically-Faster, Attribute-Based. In Advances in Cryptology – CRYPTO 2013 (pp. 75–92). Springer.

[14] Chillotti, I., Gama, N., Georgieva, M., & Izabachène, M. (2017). Faster Packed Homomorphic Operations and Efficient Circuit Bootstrapping for TFHE. In Advances in Cryptology – ASIACRYPT 2017 (pp. 377–408). Springer.

[15] Chillotti, I., Ligier, D., Orfila, J. B., & Tap, S. (2021). Improved Programmable Bootstrapping with Larger Precision and Efficient Arithmetic Circuits for TFHE. In ASIACRYPT 2021 (pp. 670–699). Springer.

[16] Cheon, J. H., Kim, A., Kim, M., & Song, Y. (2017a). Homomorphic Encryption for Arithmetic of Approximate Numbers. In Advances in Cryptology – ASIACRYPT 2017 (pp. 409–437). Springer.

[17] Iliashenko, I., & Zucca, V. (2021). Faster Homomorphic Comparison Operations for BGV and BFV. Cryptology ePrint Archive, Paper 2021/315.

[18] Rückert, M., & Schneider, M. (2010). Estimating the Security of Lattice-Based Cryptosystems. Cryptology ePrint Archive, Paper 2010/137.

[19] Bataa, M., Song, S., Park, K., Kim, M., Cheon, J. H., & Kim, S. (2020). Homomorphic Computation of Local Alignment. In 2020 IEEE International Conference on Bioinformatics and Biomedicine (BIBM) (pp. 2167–2174). IEEE.

Hussein Aghajani Kalkhouran is a PhD student in the Department of Computer Science at AGH University of Science and Technology, Krakow, Poland. His research interests include cryptography, fully homomorphic encryption, privacy-preserving computation, and their applications to secure systems. He focuses on advancing the practical deployment of FHE technologies, particularly TFHE, for real-world privacy-critical applications.