SoK: Can Fully Homomorphic Encryption Support General AI Computation? A Functional and Cost Analysis

Anonymous Author(s)

Abstract

Artificial intelligence (AI) increasingly powers sensitive applications in domains such as healthcare and finance, relying on both *linear operations* (e.g., matrix multiplications in large language models) and *non-linear operations* (e.g., sorting in retrieval-augmented generation). Fully homomorphic encryption (FHE) has emerged as a promising tool for privacy-preserving computation, but it remains unclear whether existing methods can support the full spectrum of AI workloads that combine these operations.

In this SoK, we ask: *Can FHE support general AI computation?* We provide both a functional analysis and a cost analysis. First, we categorize ten distinct FHE approaches and evaluate their ability to support general computation. We then identify three promising candidates and benchmark workloads that mix linear and non-linear operations across different bit lengths and SIMD parallelization settings. Finally, we evaluate five real-world, privacy-sensitive AI applications that instantiate these workloads. Our results quantify the costs of achieving general computation in FHE and offer practical guidance on selecting FHE methods that best fit specific AI application requirements.

Keywords

Fully Homomorphic Encryption, Generality Measurement

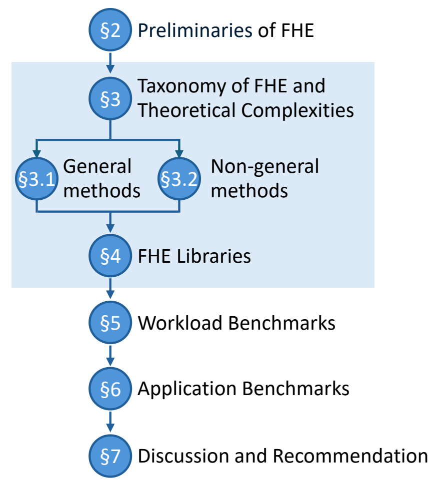

Figure 1: Overview of Our FHE Generality Measurements.

1 Introduction

The rapid advancement of AI has enabled its widespread adoption across privacy-sensitive domains, including healthcare [55, 90, 110] and finance [12, 46]. To protect privacy, FHE is a promising tool, as it enables direct computation on encrypted data without decryption, ensuring end-to-end confidentiality. However, AI workloads rely on both linear and non-linear operations, while existing FHE schemes are specialized rather than general: word-wise schemes (e.g., BGV [22], BFV [21, 39], CKKS [29]) are efficient for linear operations but struggle with non-linear ones, whereas bit-wise schemes (e.g., FHEW [38], TFHE [34]) handle non-linear operations well but incur prohibitively high costs on linear ones. Thus, it remains unclear whether existing FHE can truly support the full spectrum of AI workloads that mix linear and non-linear operations in practice. For example, neural networks [42, 77] alternate linear matrix multiplications with non-linear activation functions, and graph algorithms such as Floyd–Warshall [40] combine linear aggregation with non-linear comparisons.

We refer to such mixed computations as general computation. The ability of an FHE method to support general computation in a non-interactive and efficient manner is what we call *computational generality*. Although prior research has systematized FHE along dimensions such as performance, toolchain maturity, and developer accessibility [43, 98], computational generality has received little attention—despite being essential for real AI applications. The open research question is twofold: *first, the functionality of different FHE methods in supporting general computation, and second, the cost of executing it when such support is available.*

Given the diverse landscape of FHE methods that are specifically optimized for different applications and workloads, evaluating the FHE's computational generality is challenging. Many FHE methods are theoretically capable of general computation but remain impractical due to significant computational complexity. For instance, directly using bit-wise FHE for general computation is impractical because of its extremely high computational complexity for linear computation [79], i.e., multiplying two 16-bit integers in TFHE can take up to 30 seconds. On the other hand, using word-wise FHE for non-linear functions with linear approximation appears to be a potential direction for handling general computation, such as polynomial approximation [30, 31, 65] for CKKS. However, these methods only work on a small interval, such as $[-1, 1]$, inevitably introducing errors around 0. Such errors render approximation-based methods unsuitable for applications where even small errors are intolerable, such as genomics [55, 90, 110] and finance [12, 46]. Another line of work for general computation involves scheme switching [20, 79] between word-wise and bit-wise FHE, allowing for the evaluation of the linear operations in word-wise and the non-linear operations in the bit-wise FHE without decryption. While this leverages the strengths of both types, switching cost is prohibitively high, especially for input with large bit-width [32]. Most importantly, additional bootstrapping might be needed after

Proceedings on Privacy Enhancing Technologies YYYY(X), 1–17

https://doi.org/XXXXXXX.XXXXXXX

scheme switching [8], which is one of the most computationally expensive operations in the word-wise FHE and is often avoided in practice [52].

Additionally, while word-wise BGV/BFV can leverage $\mathbb{F}_p$-based polynomial interpolation [52, 84, 96] to enable exact non-linear operations, their efficacy in general computations remains unverified, because the inherent field disparity between $\mathbb{F}_p$ (non-linear) and $\mathbb{Z}_{p^d}$ (linear) fundamentally limits seamless integration of mixed operations. To enable them to support general computation in a non-interactive manner, Zhang et al. [108] proposed an encoding switching method, HEBridge. Recently, several functional bootstrapping methods [10, 62, 71] have been proposed to allow arbitrary function computation during the bootstrapping process. Functional bootstrapping refreshes ciphertexts and applies a chosen function simultaneously, effectively removing noise while completing a specific Look-Up Table (LUT) evaluation. Specifically, Lee et al. [62] proposed a functional bootstrapping technique that takes an RLWE ciphertext, i.e., CKKS or BFV, as input and outputs the refreshed BFV ciphertext, building upon BFV-style bootstrapping. However, such BFV-style bootstrapping is far from efficient; for instance, it can take about 3 minutes for a single evaluation due to the intrinsically inefficient slot utilization of BFV-style bootstrapping [10]. To address this issue, Alexandru et al. [10] presented a general functional bootstrapping technique based on CKKS-style bootstrapping, which has the best throughput among all FHE methods, to improve amortized performance. However, this method suffers from high computational complexity for large input spaces. Their experimental results are limited to 12-bit LUTs because the scaling factor nears the limit for 64-bit modular operations, and the complexity of polynomial evaluation increases significantly, i.e., approximately 1 minute per evaluation for 9-bit LUTs, but about 10 minutes for 12-bit LUTs.

To evaluate the computational generality of FHE, we proceed in three stages, as shown in Figure 1. First, we survey and analyze current FHE methods to determine whether they have the functionality to support general computation. For FHE methods that support general computation, we analyze their theoretical complexities and identify the three most efficient candidates. Next, we conduct microbenchmarks on three representative general workloads involving mixed linear and non-linear operations: (1) linear followed by non-linear, (2) non-linear followed by linear, and (3) a sequence of linear, non-linear, and linear operations. Finally, we extend our evaluation to five real-world, privacy-sensitive applications that inherently require FHE computational generality, including Floyd–Warshall graph analysis, decision tree inference, sorting, database aggregation, and neural network inference. These applications span domains such as finance and healthcare, where protecting sensitive information during complex computations is critical. By integrating theoretical analysis with empirical runtime measurements, we deliver a comprehensive, application-driven benchmark of general FHE solutions.

2 Preliminaries

2.1 Fully Homomorphic Encryption

Fully Homomorphic Encryption (FHE) is an encryption technique that enables computations to be carried out directly on encrypted data, ensuring confidentiality throughout processing. Formally, if $y = f(x)$ represents an arithmetic function on plaintext x, there exists a function $g(\cdot)$ operating in the encrypted domain such that $y = f(x) \Leftrightarrow y = \mathrm{Dec}(g(\mathrm{Enc}(x)))$. $\mathrm{Enc}(\cdot)$ and $\mathrm{Dec}(\cdot)$ denote encryption and decryption, respectively, while $g(\cdot)$ mirrors $f(\cdot)$ in the ciphertext space. FHE methods introduce noise into ciphertexts to ensure security, typically following the Learning With Errors (LWE) problem. The noise prevents cryptanalysis but must remain below a threshold to ensure successful decryption. The security of modern FHE methods relies on the hardness of LWE and its ring-based variant, Ring LWE (RLWE). In RLWE, the goal is to find the secret $s \in R_p$ in the equation $b = a \cdot s + e$, where $b, a \in R_p$ are known, and e is an error sampled from a distribution over R_p. The ring R_p is defined as $R_p = \mathbb{F}_p[x]/\langle x^n + 1\rangle$, with n a power of 2 and $p \equiv 1 \pmod{2n}$. In practice, ciphertexts are encrypted as tuples of polynomials in this ring, taken modulo an irreducible cyclotomic polynomial whose roots are the N^{th} primitive roots of unity. The cyclotomic order (degree of the ciphertext polynomials) is typically between 2^{10} and 2^{15}, and the coefficient modulus q is often a product of primes, reaching several hundred bits.

Most current FHE methods can be categorized according to their capability to support different types of operations. Word-wise FHE methods (such as BFV [39], BGV [22], and CKKS [29]) excel at linear operations such as matrix multiplication, while bit-wise FHE methods (such as TFHE [34] and FHEW [38]) are better suited for non-linear operations such as comparison.

Bit-wise FHE methods. The first category includes bit-wise FHE methods, such as TFHE [34] and FHEW [38], which encrypt individual bits into separate ciphertexts, allowing for a wide range of encrypted bitwise manipulations and logic gate evaluations. These schemes support the construction of Boolean circuits composed of encrypted logic gates, facilitating efficient operations directly on encrypted data. TFHE, in particular, is known for its fast bootstrapping, which is essential for evaluating homomorphic logic gates. Bit-wise FHE methods enable developers to leverage decades of digital circuit design research, enabling many applications such as privacy-preserving machine learning and relevant TFHE accelerators [47, 53, 59, 73–75, 112–114]. However, they face challenges with addition and multiplication circuits [75, 79], especially when dealing with large circuit depths and fan-in bits. For example, TFHE [34] takes about 30 seconds to multiply two encrypted 16-bit integers. Also, the expansion ratio of bit-wise FHEs is usually several orders of magnitude larger than that of word-wise FHEs, leading to higher communication costs.

Word-wise FHE methods. Word-wise FHEs can be classified into integer schemes [21, 22, 39] and floating-point schemes [29]. The two most important FHE schemes that encrypt integers modulo a user-determined modulus p are BGV [22] and BFV [39]. FHE addition and multiplication correspond to modular addition and multiplication over the plaintext values. It is also possible to emulate Boolean circuits in integer schemes by setting $p = 2$, which causes addition to behave as an `XOR` gate and multiplication to work as an `AND` gate. Unlike bit-wise FHE schemes, both BGV and BFV have considerably slower bootstrapping; depending on parameter choices (such as the ring dimension), a single bootstrapping can take anywhere from several seconds to several hours [41]. Thus, most implementations of BGV and BFV do not include bootstrapping and are used exclusively in LHE mode [43].

For the floating-point schemes, they use a plaintext type of floating-point numbers, and the most popular scheme in this category is CKKS [29]. It is desirable for certain classes of algorithms, such as machine learning [102, 109, 115]. In practice, the core operations of floating-point schemes parallel those of integer schemes, with only a few differences. Similarly, the floating-point schemes have slow bootstrapping procedures and are predominantly used in LHE mode. The key difference is the need to keep track of the *scale* factor that is multiplied by plaintext values during encoding, which determines the bit-precision associated with each ciphertext. The scale doubles when two ciphertexts are multiplied, which may result in overflow as the scale becomes exponentially larger. Thus, *rescaling* must be invoked to preserve the original scale after multiplications, which is similar to modulus switching and serves a similar role in reducing ciphertext noise.

The key advantage of word-wise FHE over bit-wise methods is its support for batching. Single Instruction Multiple Data (SIMD) enables encrypting a vector of plaintexts into a single ciphertext [29, 93]. Each plaintext occupies a *slot*, with the number of slots determined by encryption parameters. Batched ciphertexts support slot-wise addition and multiplication. A key limitation is slot dependency—if operations require interaction across slots (e.g., summing all slots), slot-wise rotations are needed, incurring significant memory and runtime overhead.

Despite their efficiency in linear operations, word-wise FHE methods struggle with non-linear functions like comparison [76, 78], prompting efforts to support such functions within word-wise schemes [52, 96]. CKKS supports approximate computation on floating-point numbers, and the most prevailing method to perform the non-linear functions is through polynomial approximation [30, 31, 65]. These methods use a composition of minimax approximation polynomials [65] to approximate the sign function with errors. These methods primarily work on only small intervals, such as $[-1, -\epsilon] \cup [\epsilon, 1]$ with errors. The smaller the errors are, the higher the degree of polynomial required. Thus, more multiplication and multiplicative depth are needed. Most importantly, the approximation can never be accurate around 0. Such errors limit the application of approximation-based non-linear where accuracy is important, such as genomics [55, 90, 110] and finance [12, 46]. On the other hand, since computations are exact in BFV and BGV, the non-linear functions can be implemented in BFV and BGV without approximation error. There has been a series of works [52, 84, 96] exploring the efficient comparison in BFV and BGV. On a high level, they compute the non-linear functions by evaluating an interpolation polynomial over the base field. However, the degree of the interpolation polynomial grows exponentially with the bit-width of the input [96] or relies on a special plaintext space [52] that does not support seamless computation of the linear and non-linear operations.

2.2 Bootstrapping Overview

Bootstrapping is a fundamental operation in HE that refreshes noisy ciphertexts to enable unbounded computation. Each homomorphic operation introduces noise into ciphertexts, and once noise exceeds the decryption threshold, correctness fails. Bootstrapping homomorphically evaluates the decryption function on a noisy ciphertext, producing a refreshed ciphertext with reduced noise: $\mathbf{ct}' = \mathrm{Enc}'_{\mathrm{sk}}(\mathrm{Dec}_{\mathrm{sk}}(\mathbf{ct})) = \mathrm{Enc}'_{\mathrm{sk}}(m)$. The three computationally general FHE methods employ fundamentally different bootstrapping strategies with dramatically different costs. Bit-wise TFHE uses gate bootstrapping (around $10-20$ ms per gate), where each gate evaluation is immediately followed by noise refresh, enabling constant multiplicative depth but requiring separate bootstrapping for each bit. Word-wise methods (Scheme Switching and Encoding Switching) rely on BGV/BFV bootstrapping, which is orders of magnitude slower (3-30 seconds per operation) but can amortize costs across thousands of SIMD slots, achieving 1-7 ms per slot when batching is possible. Due to these prohibitive word-wise bootstrapping costs, most implementations operate in leveled mode with sufficiently large parameters to avoid bootstrapping entirely.

2.3 Generality Required Privacy-sensitive Applications

Over the past decade, FHE has evolved from a purely theoretical construct into a practical solution for today's data-centric challenges [43]. In finance, Intesa Sanpaolo's collaboration with IBM leverages FHE secure asset transactions [51], allowing for both confidential database querying and the execution of complex machine learning algorithms on encrypted data. Similarly, in the medical domain, Roche's partnership with ETH investigates the use of FHE for secure graph analyses of genomic information [72, 106]. As industries increasingly adopt advanced analytics that require both linear and non-linear operations, FHE's generality to handle complex computations on encrypted data makes it an indispensable tool for ensuring data confidentiality and regulatory compliance.

In Section 3, we analyze the computational generality of various FHE methods from a theoretical perspective. Section 6 presents experimental evaluations assessing their performance across diverse workloads. These workloads include sequences of operations such as non-linear applied after linear operations, mirroring the activation functions in neural networks; as well as linear following non-linear operations, which form the basic unit of basic $\mathrm{Max}/\mathrm{Min}(A, B)$ operations, expressed as $\mathrm{Compare}(A, B) \times A + (1 - \mathrm{Compare}(A, B)) \times B$. Finally, in Section 7, we evaluate privacy-sensitive AI related applications built on these workloads, demonstrating the practical generality of FHE methods. The applications under study include graph algorithms, decision trees, database retrieval, neural-network inference, and sorting, each representing a critical function in real-world research and commercial settings.

3 FHE Generality Taxonomy and Complexity Analysis

General computation is a crucial requirement of real-world privacy-sensitive applications, yet little research has investigated how existing FHE methods support this functionality. Specifically, an AI-friendly FHE method must satisfy two conditions:

Mixed Linear and Non-Linear Operations: An FHE method with computational generality must support both linear and non-linear operations on ciphertexts and allow them to be composed in arbitrary sequences. This capability enables complex computations

Table 1: Functionality and cost comparison of representative FHE methods categorized by functional capability and theoretical complexity under b-bit precision and 2^λ security level. The last three lines represent the methods achieve full computational generality across linear and non-linear operations. N is the ring dimension; for polynomial approximation of CKKS, n denotes the polynomial degree.

Methods	Non-linear Operation (i.e., Comparison)					Linear Operation (i.e., Multiplication)				Generality
	Depth	Complexity	SIMD	Exact	Efficiency	Depth	Complexity	SIMD	Efficiency	
Word-wise BGV/BFV [21, 22]	-	-	-	-	-	1	$O(\lambda^2)$	✓	✓	✗
Word-wise CKKS [29]	-	-	-	-	-	1	$O(N \log N)$	✓	✓	✗
Poly. Approx. (CKKS) [30, 65]	$O(\log_2 n)$	$O(\sqrt{n})$	✓	✗	✓	1	$O(N \log N)$	✓	✓	✗
Poly. Interp. (BGV/BFV) [85]	$O(b)$	$O(\sqrt{2^b})$	✓	✓	✗	1	$O(\lambda^2)$	✓	✓	✗
Decomp. (BGV/BFV) [52]	$O(\log_2 b)$	$O(b \log_2 b)$	✓	✓	✓	-	-	-	-	✗
XCMP (BGV/BFV) [80]	$O(1)$	$O(1)$	✓	✓	✓	-	-	-	-	✗
General Functional Bootstrapping [10]	$O(b)$	$O(\sqrt{2^b} + b)$	✓	✓	✗	1	$O(N \log N)$	✓	✓	✗
Bit-wise TFHE [34]	$O(b)$	$O(b)$	✗	✓	✓	$O(b)$	$O(b^2 p)\,CMUX + O(bp)\,KS$	✗	✗	✓
Scheme Switching [20, 79]	-	$\Omega(2^b)$	✗	✓	✗	1	$O(\lambda^2)$	✓	✓	✓
Encoding Switching [108]	$O(\log_2 d + d \log_2 p)$	$O(d^2 \sqrt{p})$	✓	✓	✓	$O(1)$	$O(\lambda^2)$	✓	✓	✓

that alternate between operation types, such as machine learning algorithms that apply linear operations (e.g., matrix multiplications) followed by non-linear activation functions (e.g., ReLU). By seamlessly mixing linear and non-linear operations, such FHE methods can accommodate a wide range of applications.

Non-Interactive Computation: An FHE method with computational generality should support these mixed operations in a non-interactive manner, allowing computations on encrypted data without further communication between the data owner and the computing party to decrypt and re-encrypt. Non-interactivity is essential for scalability and efficiency, especially in settings where minimizing communication overhead is critical.

Among existing FHE methods, only a few can simultaneously satisfy both of above conditions that can support both linear and non-linear computation in a non-interactive manner, what we term full computational generality. This section primarily focuses on these three computationally general FHE methods: bit-wise TFHE, Scheme Switching, and Encoding Switching. We analyze their theoretical complexities and evaluate their suitability for general AI computation in Section 3.1.

Other FHE methods, such as native word-wise BGV/BFV/CKKS, polynomial approximation, polynomial interpolation, and functional bootstrapping, provide only partial or approximate forms of computational generality. We discuss these method in Section 3.2 for conceptual completeness and to contextualize our focus. Table 1 summarizes the generality functionality and computational complexities of representative FHE methods, highlighting the distinction between general and un-general general approaches.

3.1 Analysis of General FHE methods

This section analyzes three FHE methods that achieve computational generality, supporting both linear and non-linear operations in a non-interactive manner. The first is bit-wise FHE[34, 38], which inherently enables both operation types through arithmetic and boolean circuits. The second is scheme switching[20, 79], which combines bit-wise and word-wise FHE to leverage the strengths of each. The third is encoding switching (i.e., HEBridge [108]), which integrates linear and non-linear operations within word-wise BGV/BFV by bridging distinct encoding spaces through homomorphic reduction and lifting functions.

Bit-wise FHE. TFHE [34] is a bit-wise FHE method that supports gate-level operations on encrypted data, enabling fast non-linear operations due to its efficient bit-wise processing. For a ciphertext with b-bit precision, TFHE can perform non-linear functions with a depth of $O(b)$ and complexity of $O(\sqrt{2^b})$. However, TFHE's linear operations are comparatively slower due to its bit-wise design. Multiplication on ciphertexts requires a depth of $O(b^2)$, along with $O(b^2 p)$ CMUX operations and $O(bp)$ key-switching. Additionally, TFHE lacks support for batch processing, i.e., SIMD, making linear computations less efficient than in word-wise FHEs, which can parallelize linear operations.

Scheme Switching. Scheme switching [20, 79] between bit-wise and word-wise ciphertexts enables general computation by leveraging the strengths of both schemes: linear functions are evaluated in word-wise FHE, and non-linear functions in bit-wise FHE. However, the complexity of scheme switching grows at least exponentially with input bit-width [17, 91], i.e., $\Omega(2^b)$. For example, evaluating 6-bit inputs takes 43.8 seconds, while 8-bit inputs require 162.7 seconds. Consequently, scheme switching becomes impractical for larger inputs.

Encoding Switching. Prior studies [52, 96] have shown that word-wise BGV/BFV support efficient and precise non-linear operations via polynomial interpolation. However, these methods cannot be seamlessly incorporated with linear operations because the plaintext space for linear operations in FV over $\mathbb{Z}_{p^d}$ is different that used for non-linear operations in base-encoded FV (beFV) over $\mathbb{F}_p$. To enable continuous evaluation of both operation types, Zhang et al. [108] introduced conversion techniques that employ a reduction function, which homomorphically converts the plaintext space from FV to beFV, allowing non-linear operations to follow linear operations, and a lifting function that restores the result from beFV back to the original FV space for subsequent linear operations. In terms of complexity, given that the FV plaintext modulus is p^d, the reduction function requires $O(d^2 \sqrt{p})$ multiplications and $O(d \log_2 p)$ multiplicative depth, while the lifting function requires $O(\sqrt{dp})$ multiplications and a depth of $O(\log_2 d + \log_2 p)$.

3.2 Analysis of un-General FHE methods

While the three methods analyzed in Section 3.1 provide full computational generality, several other FHE approaches offer partial or approximate support for mixed linear and non-linear operations. We provide a brief overview of these methods here for conceptual completeness, emphasizing why they fall short of full generality.

Word-wise FHE Schemes. Word-wise FHEs, including BGV, BFV, and CKKS, naïvely support efficient linear operations with SIMD. However, computing non-linear functions in SIMD-enabled word-wise FHE is non-trivial. As a result, many efforts have been made to enable non-linear functions within word-wise FHE to leverage SIMD. For the CKKS scheme, the most common approach is polynomial approximation [30, 31, 65]. On the other hand, for integer-based BGV and BFV, Narumanchi et al. [85] proposed evaluating non-linear functions via polynomial interpolation over the base field.

CKKS with Polynomial Approximation. CKKS can approximate non-linear functions (e.g., sign, ReLU, sigmoid) using compositions of minimax approximation polynomials over small intervals [66, 68] such as $[-1, 1]$. Given an n-degree approximation polynomial, the complexity is $O(\sqrt{n})$ and the multiplicative depth is $\log_2 n$. Higher polynomial degrees reduce approximation error but increase computational cost and multiplicative depth.

Limitation for General Computation: Polynomial approximation introduces inherent errors, especially near zero, making it unsuitable for applications requiring exact computation, such as genomics and finance. While it provides a form of computational generality, it is approximate rather than exact, limiting its applicability in error-sensitive domains. More details are in the Appendix A.1.

BGB/BFV with Polynomial Interpolation. In contrast to CKKS, computations in BGV/BFV are exact. Non-linear functions can be implemented via polynomial interpolation over the base field $\mathbb{F}_p$. For example, comparison operations can be expressed using Lagrange interpolation polynomials evaluated homomorphically on encrypted differences [85].

Limitation for General Computation: While polynomial interpolation supports exact non-linear operations, the degree of the interpolation polynomial grows exponentially with input bit-width when performed over $\mathbb{Z}_{p^r}$. To maintain efficiency, operations are typically confined to small primes $p \leq 257$, making this approach impractical for large inputs. Additionally, the plaintext space for linear operations ($\mathbb{Z}_{p^d}$) differs from that used for non-linear operations $\mathbb{F}_p$, preventing seamless composition without encoding conversion (as in Encoding Switching [108]). More details are in the Appendix A.2.

Polynomial Interpolation with Special Encoding. To enhance the scalability of polynomial interpolation, special encoding methods [52, 96] decompose large integers into vectors of base-p digits. Non-linear operations (e.g., comparisons) are then performed digit-wise, reducing depth from $\log_2 p^r$ to $\log_2 \log_p 2^b + \log_2(p-1) + 4$.

Limitation for General Computation: While these methods significantly improve the efficiency of non-linear operations, the specialized ciphertext format (vector encoding) does not support standard linear operations. Thus, they cannot seamlessly compose linear and non-linear operations without format conversion, disqualifying them as fully general FHE methods. More details are in the Appendix A.3.

Exponential Encoding (XCMP). XCMP [80] uses exponential encoding to perform private comparisons with constant multiplicative depth $O(1)$ by encoding values as polynomial degrees. This enables highly efficient comparisons but is limited to small input domains (typically < 16 bits) due to the polynomial degree constraint.

Limitation for General Computation: XCMP's specialized encoding precludes direct composition with linear operations, and extensions to larger domains incur prohibitive multiplicative depth. More details are in the Appendix A.4.

General Functional Bootstrapping. Functional bootstrapping [10, 62, 71] extends traditional bootstrapping by enabling the evaluation of arbitrary functions (via look-up tables) during noise refresh. Methods such as BFV-style [62] and CKKS-style [10] functional bootstrapping approximate target functions using interpolation polynomials with complexity roughly proportional to $\sqrt{p}$ for first-order interpolation.

Limitation for General Computation: While theoretically powerful, functional bootstrapping remains practical only for small input spaces ($\leq$ 12 bits) due to polynomial degree growth and scaling factor constraints. Moreover, extending to larger inputs requires digit-wise decomposition (similar to special encoding methods), which again precludes seamless linear-nonlinear composition. More details are in the Appendix A.5.

4 Relationship to Prior SoK work

Gouert et al. [43] presented a comprehensive SoK on FHE libraries, providing standardized benchmarks across major implementations including HElib, SEAL, Lattigo, OpenFHE, TFHE, and Concrete. Their work focuses on library-level performance evaluation—measuring runtime, memory consumption, ciphertext expansion, and parameter selection workflows across standard primitive operations (encryption, addition, multiplication, rotation). This library-focused SoK enables practitioners to compare implementation quality, identify performance bottlenecks, and select appropriate libraries for deployment. Additionally, Viand et al. [98] primarily surveys FHE compilers, emphasizing performance, toolchain maturity, and developer accessibility. In contrast, our SoK addresses a fundamentally different research question: "Can existing FHE methods support general AI computation?" Rather than benchmarking library implementations, we analyze the functional generality and computational scalability of underlying FHE methods when applied to AI workloads requiring both linear and non-linear operations. Table 2 summarizes the key distinctions between these three complementary systematization efforts.

Table 2: Comparison of FHE SoK contributions.

SoK	Focus	Units Evaluated	Contribution
Viand et al. [98]	Compiler usability & performance	Compilers (EVA, CHET, Cingulata, etc.)	Compiler landscape systematization
Gouert et al. [43]	Library performance	Libraries (HElib, SEAL, Lattigo, TFHE, PALISADE)	Standardized library benchmarking
Our Work	**Computational generality**	**FHE Methods (TFHE, Scheme Switching, Encoding Switching)**	**Method selection for general AI**

5 FHE Libraries Supporting General Computation

Various open-source FHE libraries implement the aforementioned FHE methods and provide high-level APIs that allow users to select encryption parameters tailored to their applications. In this section, we list six widely used libraries and summarize their support for these FHE methods.

Table 3 summarizes FHE libraries alongside the encryption schemes they implement and the types of supported operations. In general, HELib, Lattigo, and SEAL offer word-wise FHEs by supporting BFV, BGV and CKKS. In contrast, TFHE and Zama's TFHE-rs are designed primarily for bit-wise FHEs, offering Boolean gate-level primitives that can be composed into more complex functions. Notably, OpenFHE and Zama extend this generality further by facilitating Scheme Switching or hybrid approaches that bridge bit-wise and word-wise computations. Additionally, methods Polynomial Interpolation [85] and Polynomial Interpolation with Special Encoding [52] are open-sourced based on HELib, while Encoding Switching [108] is open-sourced using HELib. Functional bootstrapping techniques [10, 62] are implemented on Lattigo and OpenFHE but have not yet been open-sourced.

Table 3: Compatibility of libraries with various FHE methods.

Methods	HElib	Lattigo	SEAL	TFHE	OpenFHE	Zama
Word-wise BGV	✓	✓	✓	✗	✓	✗
Word-wise BFV	✗	✓	✓	✗	✓	✗
Word-wise CKKS	✓	✓	✓	✗	✓	✗
Bit-wise TFHE	✗	✗	✗	✓	✓	✓
Poly. Interp.	✓	✗	✗	✗	✗	✗
Poly. Approx.	✓	✓	✓	✗	✓	✗
Scheme Switching	✗	✓	✗	✗	✓	✗
Encoding Switching	✓	✗	✗	✗	✗	✗
General Functional Bootstrapping	✗	✗	✗	✗	✗	✗

HELib. The Homomorphic Encryption Library (HELib) was introduced in 2013 by IBM and supports the BGV scheme (with bootstrapping), as well as CKKS. Polynomial Interpolation [52, 85] with BGV and Encoding Switching [108] are implemented on HELib and have been open-sourced. HELib is written in C++17 and uses the NTL mathematical library.
Lattigo. The lattice-based multiparty HE library in Go was first developed by the Laboratory for Data Security (LDS) at EPFL and is currently maintained by Tune Insight. It supports BFV, BGV, and CKKS. Lattigo enables scheme switching to compute non-linear functions. Functional FV bootstrapping [62] is built on the lattigo, but their implementation is not open-sourced yet.
SEAL. The Simple Encrypted Arithmetic Library (SEAL) was developed by Microsoft Research and was first released in 2015 [3]. SEAL supports leveled BFV, BGV, and CKKS.
TFHE. The Fast Fully Homomorphic Encryption Library over the Torus (TFHE) was released in 2016 by Chillotti et al. [34] and proposes the CGGI cryptosystem. The library exposes homomorphic Boolean gates such as AND and XOR but does not build complex functional units (e.g., adders, multipliers, and comparators) and leaves that to the developer.
OpenFHE. OpenFHE [8] is developed by Duality, NJIT, MIT, and other organizations. It supports a wide range of FHE methods, including BGV, BFV, and CKKS with approximate bootstrapping, as well as DM/FHEW, CGGI/TFHE, and LMKCDEY for evaluating Boolean circuits. OpenFHE enables scheme switching between CKKS and between CKKS and FHEW/TFHE to evaluate non-smooth functions, e.g., comparison, using FHEW/TFHE functional bootstrapping. Recently, Alexandru et al. [10] leveraged OpenFHE to build general functional bootstrapping to enable the non-linear function for any RLWE ciphertexts, but it is not open-sourced yet.
Zama. Zama [103] is an open-source cryptography company developing state-of-the-art FHE solutions for blockchain and AI. Its products include TFHE-rs [107] for Boolean and small integer arithmetic, Concrete [105] for compiling Python to FHE with LLVM, Concrete ML [104] for encrypted machine learning, and fhEVM for confidential smart contracts in Solidity.

6 General Computation Required Workloads

6.1 Design Principle

To comprehensively evaluate and compare different general FHE methods, we designed three workloads requiring generality, each representing basic computational units in privacy-sensitive AI applications, covering diverse scenarios:

- Workload-1: Compare $(\mathrm{Enc}(A) \times \mathrm{Enc}(B), \mathrm{Enc}(C))$, a non-linear operation following a linear operation that serves as a basic unit for database queries. For instance, it can be used to determine if the product of two encrypted values exceeds a given threshold.
- Workload-2: Compare $(\mathrm{Enc}(A), \mathrm{Enc}(B)) \times \mathrm{Enc}(C)$, a linear operation following a non-linear operation that forms a basic unit in decision tree algorithms. For example, this unit can evaluate a comparison between two encrypted feature values and then multiply the result by another encrypted value to determine the weight or selection criteria at a decision node.
- Workload-3: Compare $(\mathrm{Enc}(A) \times \mathrm{Enc}(B), \mathrm{Enc}(C)) \times \mathrm{Enc}(D)$, a composite sequence of linear, non-linear, and linear operations. It is a basic component of a neural network, where it can model the execution of a convolution layer, followed by a ReLU activation, and then another convolution layer.

6.2 Experimental Setup

To evaluate the performance of general FHE methods, we conducted experiments on the above workloads and measured both the total execution time and the amortized time for SIMD-enabled word-wise Scheme switching and Encoding switching.
System Setup. The experiments are conducted on a server equipped with an AMD Ryzen Threadripper PRO 3955WX (2.2 GHz) and 125 GB of RAM. All tests are run in single-thread mode for a fair comparison. For Encoding Switching, we use the official open-sourced implementation from HEBridge [108] built on HELib. Scheme Switching is adopted from OpenPEGASUS [79] built on Microsoft SEAL, and TFHE-rs [107] are used for the bit-wise TFHE method. All encryption parameters are configured to maintain a security level above 128 bits [19, 60], following the "BKZ-beta" classical cost model from the LWE estimator [9] unless stated otherwise.

Parameter Setup. Parameter choices ensure operations remain within the multiplicative depth budget, avoiding word-wise bootstrapping. For both BGV/BFV-based methods, i.e., Encoding Switching and Scheme Switching, we set the secret distribution to be a Hamming weight distribution over the set of ternary polynomials with coefficients in $\{0, 1, -1\}$, such that each secret has exactly $h = 64$ nonzero entries.

For Encoding Switching, which is built on HELib, the plaintext modulus in the FV space is given by p^r. We choose the parameters p and r based on the input bit-width. Specifically, we use the pairs $\{(4, 4), (5, 4), (7, 5), (17, 4)\}$ as (p, r) for input bit-widths of 6, 8, 12, and 16, respectively. The multiplicative depth is determined by the ciphertext capacity, defined as $\log_2 \frac{q}{\eta}$, where q is the ciphertext modulus and η is the current noise bound. Accordingly, we set $\log_2 q$ to $\{256, 320, 488, 648\}$ for 6, 8, 12, and 16 bits of input. After choosing p, r, and q, we select an appropriate degree for the polynomial modulus to ensure that the security level satisfies $\lambda > 128$. In this context, the cyclotomic order of the polynomial ring m is chosen so that the ring degree n and the order of the base prime, $d = \texttt{Ord}(p)$, meet the following values: $\{(m, n) = (13201, 12852), (16151, 15600), (25301, 25300), (31621, 31212)\}$, corresponding to 6, 8, 12, and 16 bits of input, respectively.

For Scheme Switching, which is implemented using Microsoft SEAL, we set the parameters in a way that mimics the above setup to ensure a fair comparison. Specifically, we explicitly configure the plaintext modulus p^r using the identical (p, r) pairs, and we select the ciphertext modulus q with the same $\log_2 q$. Then, we choose a polynomial modulus degree n (which in SEAL must be a power of two) such that the overall security level satisfies $\lambda > 128$. Although Microsoft SEAL does not allow explicit setting of the cyclotomic order m and the order of p (denoted as d), these are implicitly determined by our choices of n and p (with m being typically interpreted as $2n$ for the cyclotomic polynomial $x^n + 1$, and $d = \texttt{Ord}(p)$ accordingly).

For the bit-wise TFHE method, TFHE-rs provides `FheUint6`, `FheUint8`, `FheUint12`, `FheUint16`, `FheUint24`, and `FheUint32` to represent differnt input bit precision.

Our benchmarks focus exclusively on server-side FHE operations, including all evaluation tasks except key generation and encryption/decryption, as these are one-time user costs. No custom parallelization is applied to HElib, OpenFHE, or TFHE-rs, as each defaults to single-core execution.

6.3 Experimental Results

Figure 2(a) reports the latency of a single run, whereas Figure2 (b) illustrates the amortized time across slots for FHE methods supporting SIMD parallelism (Scheme Switching and Encoding Switching).

TFHE [34] evaluates all operations bit-wise, handling non-linear functions efficiently but performing poorly on linear ones at larger bit lengths. It lacks SIMD support, unlike word-wise schemes that process multiple ciphertexts simultaneously. Notably, TFHE performs better on workload-2 than workload-1 because, in workload-2, one multiplication input is the comparison result (all 0s or all 1s), reducing circuit depth compared to workload-1, where multiplication precedes comparison

Scheme Switching [20], on the other hand, evaluates linear operations using word-wise FHE and non-linear operations using bit-wise FHE. However, the switching process's complexity grows exponentially with the input bit-width. For instance, on workload-1, 6-bit inputs take 9.87 seconds and 8-bit inputs 32.1 seconds, with runtimes for 12- and 16-bit inputs becoming prohibitive, making the approach impractical for larger inputs.

Encoding Switching [108] avoids the overhead of expensive scheme switching and leverages SIMD capabilities by performing both linear and non-linear operations in word-wise FHE. For 8-bit inputs, workload-1 takes only 15.5 seconds, and the method scales efficiently to 12- and 16-bit inputs (23.0 and 46.9 seconds), achieving up to 14.1× and 25.4× speedups over TFHE and scheme switching, respectively.

When comparing the amortized time in Figure 2 (b) with the single-run results, TFHE overtakes scheme switching as the slowest method, owing to its lack of SIMD support.

Table 4 reports storage and communication metrics for the three general FHE methods on Workload-1 (1×100 vector of 8-bit integers). TFHE achieves the smallest footprint (4.8 MB ciphertext, 9.6 MB communication, 23.3 MB memory), making it ideal for bandwidth-constrained scenarios, while Encoding Switching exhibits 10× larger overhead (47.6 MB, 95.2 MB, 94.3 MB) but enables faster computation in resource-rich environments. These metrics introduce deployment constraints into method selection: applications with limited bandwidth or storage should prioritize TFHE despite higher computational costs, whereas datacenter deployments can accept Encoding Switching's larger footprint for performance gains.

Table 4: Storage and communication overhead (Workload-1).

Method	Ciphertext	Peak Memory	Communication*
TFHE	4.8 MB	23.3 MB	9.6 MB
Scheme Switching	14.2 MB	58.6 MB	28.4 MB
Encoding Switching	47.6 MB	94.3 MB	95.2 MB

*Communication between server and client.

7 General Computation Required Applications

7.1 Design Principle

In this section, we evaluate general FHE methods on five representative privacy-sensitive AI applications that require general computation: private graph algorithms, private decision tree inference, private database aggregation, private neural network inference, and private sorting. These applications cover a diverse range of real-world use cases across domains such as healthcare, finance, and other industrial sectors. Each application incorporates both linear and non-linear operations and is built on the workloads analyzed in Section 6.

7.2 Experimental Setup

Generally, we evaluate various applications using Scheme Switching, Encoding Switching, and bit-wise TFHE, with experimental setup details identical to those described in Section 6.2. For private graph evaluation, a graph with n nodes is represented as an $n \times n$

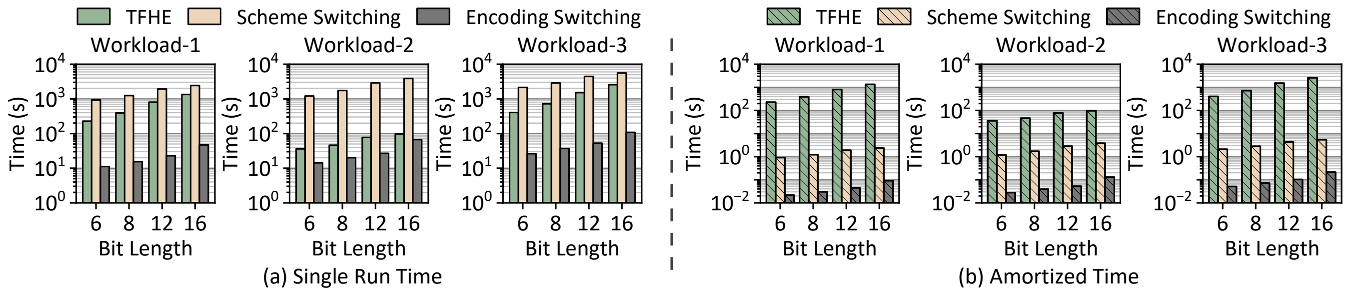

Figure 2: Comparisons on three workloads.

adjacency matrix. In the case of bit-wise TFHE, each element of the matrix is individually encrypted into multiple ciphertexts (for multiple bits), whereas for SIMD-enabled methods, each row is batched into a single ciphertext. For private decision tree evaluation, each node is encrypted as multiple ciphertexts for bit-wise FHE, while for SIMD-enabled methods, an entire layer is batched into one ciphertext. In private sorting, each element is encrypted into a single ciphertext as described in [52]. For private database aggregation, every element is encrypted into multiple ciphertexts when using bit-wise TFHE, whereas for SIMD-enabled methods, each column is batched into a single ciphertext. Finally, for private neural network inference, each parameter is encrypted into multiple ciphertexts with bit-wise TFHE, while for SIMD-enabled methods, each filter or weight matrix is batched into a single ciphertext.

7.3 Private Graph

Private graph algorithms [37, 70, 81, 99] enable secure computations on graph-structured data while preserving individual privacy [6, 82, 99]. In this threat model, an untrusted server performs computations without learning any details about the input graph. A common operation in graph algorithms is selecting the shorter path from paths P_1 and P_2, which can be implemented as: $(P_1 > P_2) \times P_2 + (1 - (P_1 > P_2)) \times P_1$; this computation exemplifies workload-2, where a linear operation follows a non-linear operation. In this section, we demonstrate the performance of general FHE methods on the private Floyd-Warshall algorithm [40].

Floyd-Warshall on Plaintext. The Floyd-Warshall algorithm is a classic algorithm for finding the shortest paths between all pairs of nodes in a graph G. It returns on a distance matrix D, where $D[i, j]$ represents the shortest known distance from node i to node j.

As shown in Algorithm 1, the algorithm iteratively updates D by considering each node as an intermediate node in potential paths. During each iteration, it updates the distance $D[i, j]$ for each pair of nodes (i, j) by comparing it with $D[i, k] + D[k, j]$, which represents a path from i to j via k. This process is repeated for all nodes k, ensuring all potential intermediate nodes are considered. After all iterations, the D contains the shortest paths between all pairs of nodes in the graph. The algorithm has a time complexity of $O(|V|^3)$, where $|V|$ is the nodes number.

Bit-wise FHE Private Floyd-Warshall. To implement a private Floyd-Warshall algorithm, a naive method [37] encrypts each element of the adjacency matrix G to ciphertext. The server then conducts the same procedure on the encrypted G as delineated in

Algorithm 1 Floyd-Warshall Algorithm on Plaintext

```
1:  Input: G: adjacency matrix
2:  Output: D: matrix of shortest path distances between each pair of nodes, P: matrix
    of predecessors for each node
3:  # Initialize distance array D and predecessor array P
4:  for each vertex v in G do
5:      D[v] = G[v], P[v] = null
6:  # Iteratively update the distances
7:  for each vertex k in G do
8:      for each vertex i in G do
9:          for each vertex j in G do
10:             if D[i,k] + D[k,j] < D[i,j] then
11:                 D[i,j] = D[i,k] + D[k,j]
12:                 P[i,j] = P[k,j]
13: return D, P
```

Algorithm 1. For if-else operations based on comparison results, the server uses HE multiplication to implement the conditional logic. Specifically, lines 10-11 of Algorithm 1 can be computed in HE as:

$$D[i, j] = M \cdot (D[i, k] + D[k, j]) + (1 - M) \cdot D[i, j], \quad (1)$$

where M is the encrypted comparison result. This naive private Floyd-Warshall requires $|V|^3$ private comparisons and $4|V|^3$ private multiplications, resulting in significant computational overhead.

Word-wise FHE Private Floyd-Warshall with SIMD. The cubic complexity of bit-wise naive private Floyd-Warshall stems from the three nested loops. However, we can optimize this process by leveraging the SIMD mechanism of word-wise FHE with the algorithm's inherent parallelism.

After fixing the middle node k, updates for paths from node i to other nodes via k can be computed simultaneously. This parallelism allows SIMD by encoding the adjacency matrix row by row.

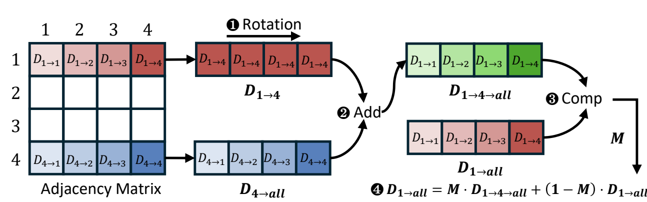

Figure 3: Toy example of SIMD update distances between node 1 to others through node 4.

Figure 3 illustrates this SIMD optimization for a graph with 4 nodes, focusing on updating distances from node 1 to all others through node 4. The process is: ❶ Generate ciphertext $D_{1\to 4}$ by multiplying $[D_{1\to1}, D_{1\to2}, D_{1\to3}, D_{1\to4}]$ with plaintext $[0, 0, 0, 1]$, then replicating $D_{1\to4}$ across all slots by rotations and addition. ❷ Compute $\boldsymbol{D_{1\to4\to all}}$ by adding $\boldsymbol{D_{1\to4}}$ to $\boldsymbol{D_{4\to all}}$, representing new potential distances via node 4. ❸ Compare $\boldsymbol{D_{1\to4\to all}}$ and $\boldsymbol{D_{1\to all}}$, yielding mask $\boldsymbol{M}$ (1 where new distance is shorter, 0 otherwise). ❹ Update distances by:

$$\boldsymbol{D_{1\to all}} = \boldsymbol{M} \cdot \boldsymbol{D_{1\to4\to all}} + (1 - \boldsymbol{M}) \cdot \boldsymbol{D_{1\to all}}, \tag{2}$$

only paths via node 4 is shorter than original one will be updated.

This SIMD-enabled approach, as shown in Algorithm 2, allows each row of the adjacency matrix to be encrypted as a single ciphertext. Consequently, paths from node i to all other nodes via node k can be compared in a single operation, reducing the total number of HE comparisons from $O(|V|^3)$ to $O(|V|^2)$, and HE multiplications from $O(4|V|^3)$ to $O(4|V|^2)$.

Algorithm 2 Floyd-Warshall Algorithm on Ciphertext Using SIMD

```
Input: G: adjacency matrix (encrypted row-by-row)
Output: D: matrix of shortest path distances between each pair of nodes (encrypted), P: matrix of predecessors for each node (encrypted)
# Initialize distance array D and predecessor array P
for each vertex v in G do
    D[v] = G[v], P[v] = null
# Iteratively update the distances using SIMD
for each vertex k in G do
    for each vertex i in G do
        D_{i→k} = SIMD(D[i, k]) # Broadcast D[i, k] to all slots
        D_{k→all} = D[k, :]
        D_new = D_{i→k} + D_{k→all}
        M = D_new < D[i, :] # Mask for minimum
        D[i, :] = M · D_new + (1 − M) · D[i, :]
        P[i, :] = M · P[k, :] + (1 − M) · P[i, :]
return D, P
```

Experimental Analysis. To assess the performance of different general FHE methods for the Private Floyd-Warshall algorithm, we conducted experiments across various input bit widths and graph sizes. Figure 4 (a) compares the performance for a graph with 32 nodes under different input bit lengths, where Encoding Switching exhibits the best performance and Scheme Switching the worst, particularly for larger bit widths. Figure 4 (b) shows results for graphs of varying sizes using an 8-bit input. As noted earlier, the runtime of bit-wise TFHE grows approximately as $O(|V|^3)$ due to the lack of SIMD support, while the complexities for both Scheme Switching and Encoding Switching are reduced to roughly $O(|V|^2)$. For 8-bit inputs, TFHE takes 14.69 seconds, and Encoding Switching takes 17.28 seconds on a 16-node graph; however, for a 128-node graph, TFHE's runtime escalates dramatically to 5,432 seconds, whereas Encoding Switching completes in 632 seconds.

7.4 Private Decision Tree Evaluation

Private Decision Tree Evaluation (PDTE) [5, 7, 14, 56] allows a server to provide predictions using a private decision tree on a client's confidential input, preserving both input privacy (the server learns nothing about the client's data) and model privacy (the client only obtains the inference result without learning any details of the decision tree).

Decision trees primarily involve comparisons and traversals [94]. In these trees, comparisons are made between the client's inputs and decision thresholds, yielding ciphertext values $c_i \in [0, 1]$ at

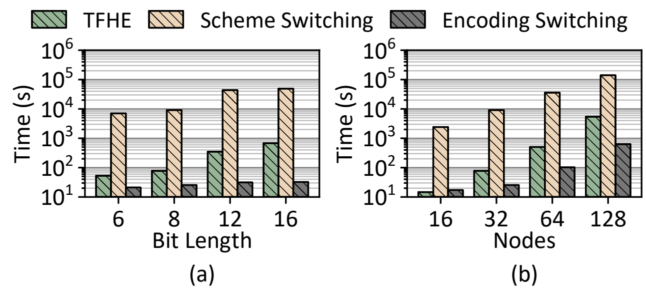

Figure 4: Homomorphic Floyd-Warshall. (a) Execution time for a 32-node graph with varying input bit lengths. (b) Execution time for graphs of different sizes with 8-bit input.

each node, while traversals determine the active path by computing the product $c_1 \cdot c_2 \cdot \dots \cdot c_d$, where c_i is the comparison result at the i-th level and d is the tree's depth. The correct decision is identified by the leaf node that decrypts to 1, with all other leaves decrypting to 0. Since private decision tree evaluation involves both linear and non-linear operations, it requires general FHE methods. Specifically, linear operations following non-linear comparisons correspond to workload-2 defined in Section 6. Notably, the traversal operation $c_1 \cdot c_2 \cdot \dots \cdot c_d$ can be reformulated [7] as:

$$c_1 \cdot c_2 \cdot \dots \cdot c_d = \overline{\overline{c_1} + \overline{c_2} + \dots + \overline{c_d}} \tag{3}$$

which significantly reduces the required multiplicative depth, further improving the efficiency, especially for decision trees with larger depths.

Experimental Analysis. In Figure 5 (a), we evaluate the performance of various general FHE methods on a complete decision tree with a depth of 6 across different input bit lengths. Figure 5 (b) shows the performance for complete decision trees of varying depths using an 8-bit input. The results indicate that for deeper trees, the word-wise Encoding Switching method performs more efficiently than others since it enables SIMD so that it can compare the same input with multiple thresholds simultaneously while avoiding the significant cost of scheme switching between bit-wise and word-wise ciphertexts. On the other hand, the word-wise Scheme Switching performs worst because of the significant frequency of switching, since every non-linear operation follows a linear operation.

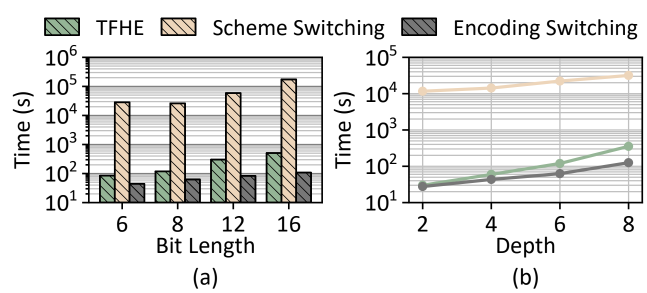

Figure 5: Private Decision Tree. (a) Execution time of inference for a depth-6 private decision tree under varying input bit precision; (b) Execution time of private decision trees with different depths using 8-bit input.

7.5 Private Sorting

Private sorting is a protocol that enables secure ordering of encrypted data without revealing the underlying values or the sorting process. In this scenario, a client encrypts its data and sends to a server, which then performs the sorting directly on the ciphertexts. Private sorting is critical and commonly used in privacy sensitive area such as finacial. For instance, several financial institutions might want to merge and sort transaction data to detect market trends without exposing individual client details.

The most efficient homomorphic sorting algorithm in terms of runing time is the direct sorting algorithm proposed by Çetin, et al. [25]. This approach directly determines final position of each element by counting the number of elements less than it. Specifically, for an array of size m, the server performs $m(m-1)/2$ homomorphic less-than operations and an equal number of homomorphic additions on the less-than operations' result for computing the Hamming weight, which yields the correct indices I for each element.

Once the indices are established, the elements are repositioned in one efficient step using a conditional multiplexer (CMUX)-like operation, as follows:

$$S[i] = \sum_{j=1}^{m} X[j] \cdot \mathrm{EQ}(i, I[j]), \tag{4}$$

where $S[i]$ is the element at the position I of sorted array S and X is the input array. $\mathrm{EQ}(i, I[j])$ is 1 if the i equals the index $I[i]$ corresponding to $X[j]$, and 0 otherwise. Constructing one position $S[i]$ requires m equality evaluation and m multiplication, so sorting the entire encrypted array requires $O(m^2)$ non-linear operations (less-than and equality evaluation) and $O(m^2)$ linear operations (addtions and multiplications).

Thus, homomorphically sorting an encrypted array of m elements demands general FHE methods, particularly as the linear operations are performed on the results of non-linear operations, corresponding to our workload-2.

Figure 6 shows an example to private sort a three-dim array $X = [a, b, c]$, the server first computes a less-than matrix L by:

$$\mathbf{L}_{ij} = \begin{cases} \mathrm{LT}(x_i, x_j) & \text{if } i < j, \\ 0 & \text{if } i = j, \\ 1 - \mathrm{LT}(x_j, x_i) & \text{if } i > j. \end{cases} \tag{5}$$

Next, the server computes the Hamming weight of each row, i.e., i-th row, to obtain how many elements are larger than x_i. The Hamming weight, i.e., $I[a]$, $I[b]$ and $I[c]$, denote the indices of a, b and c in the sorted array, respectively. Finally, the sorted array S is constructed by selecting elements $X[j]$ where the index matches the hamming weight, with Equation 4.

Experimental Analysis. We conduct experiments using various general FHE methods on the private sorting tasks. Figure 7 (a) presents the performance of different FHE methods under different input bit precisions, and Figure 7 (b) shows the performance under different array dimensions. For instance, sorting an 8-element array requires 59.9 seconds with TFHE, compared to 645 seconds for Encoding Switching and 27,457 seconds for Scheme Switching. This efficiency advantage is due to two key factors. First, unlike other applications, private direct sorting on a single array cannot leverage SIMD [52], which diminishes the benefits of word-wise methods like Scheme Switching and Encoding Switching. Second, in this context, one of the operands in the linear operations is the output of a non-linear operation (either 0 or 1), resulting in a simpler circuit compared to linear operations on arbitrary integers. Conversely, Scheme Switching shows the lowest efficiency because its frequent switching, triggered by each non-linear operation following a linear one, adds significant overhead.

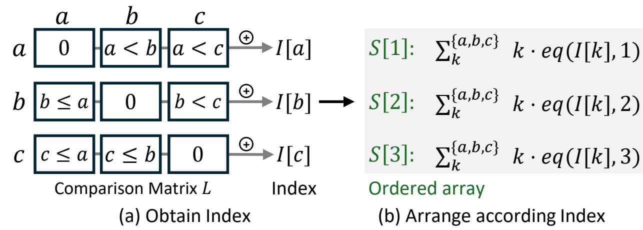

Figure 6: Toy example of direct sorting on the array $[a, b, c]$. (a) demonstrates the computation of the Comparison Matrix L and the derivation of indices from the hamming weight of L, where each element's index corresponds to the count of elements greater than it. (b) shows the sorting mechanism where each element is multiplied by a binary value: only the element with a matching index retains its value (multiplied by 1), while others are set to zero (multiplied by 0).

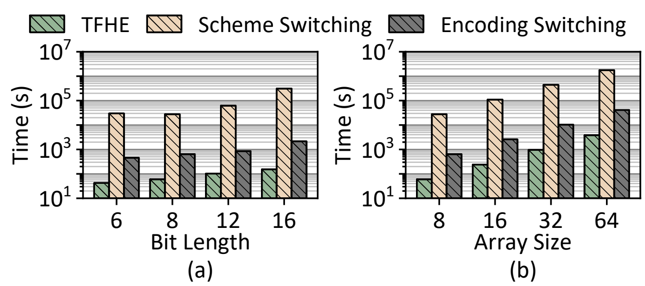

Figure 7: Homomorphic Sorting. (a) Execution time of sorting an 8-element array under varying input bit precision; (b) Execution time of sorting arrays of different lengths with 8-bit input.

The reasons are two-fold; firstly, unlike other applications, private direct sorting an encrypted array cannot leverage the SIMD [52], which disables the advantages of word-wise Scheme Switching and Encoding Switching. Secondly, one of the linear operation operands is the result of non-linear operation, i.e., 0 or 1. So that the TFHE circuit would be much less than linear operation between two random nature integers. On the other hand, Scheme Switching presents the lowest efficiency across all input bits and array sizes because of the frequent switching since every non-linear operation will follow a linear operation.

7.6 Private Database Aggregation

Private Database Aggregation (PDBA) is a protocol that protects client privacy during aggregate queries on cloud-stored databases, such as Microsoft Azure SQL Server [2] and AWS Aurora [1]. Recently, Leidos has partnered with AWS to explore the solution of using FHE to protect database [13]. It allows clients to securely store encrypted data on a cloud server and perform operations, like summation, without disclosing data or the details of the queries. The server processes these functions on encrypted data and returns only the encrypted aggregate results, ensuring the server does not access actual data or query specifics.

Recently, researchers have developed encrypted databases to ensure data privacy while enabling encrypted data processing. Cryptography-based solutions [44, 88, 89, 97], utilize cryptographic primitives such as DET [15], SE [24, 35], partially PHE [86, 92] and order OPE/ORE [18, 27]. Recent advancements have explored the application of FHE in HEDA [91], HE3DB [17], ArcEDB [111] and Engorgio [16].

An aggregation query typically involves a combination of linear operations and non-linear operations. Consider the following SQL query as an example:

```
SELECT ID FROM emp
    WHERE salary * work_hours BETWEEN 5000 AND 6000
          AND salary + bonus BETWEEN 700 AND 800;
```

Listing 1: SQL Query for Employee Salaries.

In this query, the filtering predicates "salary * work_hours BETWEEN 5000 AND 6000" and "salary + bonus BETWEEN 700 AND 800" involve a non-linear operation following a linear operation. This requirement necessitates general FHE methods and aligns with workload-1 defined in Section 6.

While word-wise FHE methods work well for linear operations, they are less effective for non-linear operations like comparisons and logic filtering. To address this limitation and leverage the high efficiency of SIMD, Ren et al. introduce a novel framework HEDA [91]. The core idea is to utilize scheme switching between RLWE and LWE ciphertexts. This approach conducts non-linear operations on LWE ciphertexts, where such operations are more naturally supported. Then, it converts the results back to RLWE ciphertexts to perform linear operations in a SIMD fashion.

Experimental Analysis. As described in Section 7.6, SQL queries involve combinations of non-linear and linear operations. To evaluate the performance of each general FHE method on the private database query, we conducted experiments on Query 1 across databases with different numbers of rows under 8-bit precision, as illustrated in Figure 8 (b).

The results show that the word-wise general FHE methods generally outperform the bit-wise TFHE, largely due to their use of SIMD capabilities, a feature that the bit-wise TFHE lacks. Furthermore, as the number of rows increases, Encoding Switching demonstrates better efficiency than Scheme Switching. For example, Encoding Switching takes 115 seconds for a 512-row database, compared to 1,626 seconds for Scheme Switching. This advantage is attributed to Encoding Switching's ability to perform non-linear operations in a SIMD-style, while Scheme Switching processes non-linear operations on bit-wise ciphertexts and cannot leverage SIMD.

7.7 Private Neural Network

One of the most valuable applications of FHE is in facilitating privacy-preserving deep learning, i.e., Deep Learning as a Service

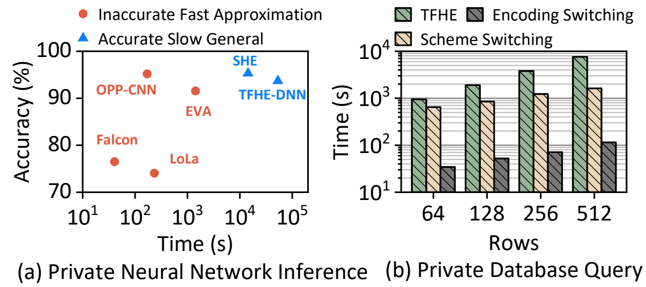

Figure 8: (a) compares the performance of various FHE DNN works on CIFAR-10 using VGG-9. (b) Private database implemented by different schemes.

(DLaaS) [83, 95], which is commonly used in information sensitive areas such as finacial [46] and healthcare [55]. This scenario involves a server that possesses a deep learning model, i.e., neural network, and a client who encrypts their input data before sharing with the untrusted server. The server processes encrypted input to provide predictions without accessing any sensitive information, thereby ensuring a secure inference service.

Neural networks consist of linear layers such as convolutions and non-linear layers like ReLU and MaxPool [58, 61]. The combination of linear and non-linear operations required by these layers necessitates general FHE methods; indeed, the iterative sequence of linear and non-linear layers forms the basis of workload-3 as defined in Section 6.

Despite the use of general FHE methods, some research leveraged the specific feature of neural network to improve efficiency but sacrificing some accuracy. Specifically, polynomial approximation-based methods [63, 66, 68] leverage the noise resilience of neural networks to enhance efficiency while reducing multiplicative depth and complexity. Although these approximations are typically accurate only within a narrow range (often between -1 and 1), activation layer inputs are usually constrained (e.g., within $[-32, 32]$ or $[-64, 64]$). By scaling these inputs to fit within $[-1, 1]$ using a predetermined scalar, servers can effectively implement these approximation techniques throughout the network, as demonstrated in approaches like DaCapo [33], OPT-CNN [54], and Hetal [69] among others [36, 64]. However, determining the appropriate scaling factor is critical: if it is too large, inputs may not be sufficiently constrained within $[-1, 1]$; if too small, the inputs become overly diminished, resulting in significant approximation errors near 0.

Another approximation strategy simulates non-linear activation functions using linear operations. For example, CryptoNets [42] replaces ReLU with the square function, a method later adopted by frameworks like LoLa [23] and Falcon [76]. This approach avoids the overhead of homomorphic non-linear operations by training the neural network with a substitute activation function.

Experimental Analysis. To evaluate the performance of various private neural network inference implementations described in Section 7.7, we conducted experiments using the CIFAR-10 dataset [57] on a VGG-9 model. VGG-9 is a streamlined version of the VGG network family, comprising 9 layers including convolutional and fully connected layers. Typically, it consists of 6 convolutional layers followed by 3 fully connected layers, with ReLU activations and max pooling operations interleaved between the convolutional layers. Encryption parameters were set as specified in each referenced paper to ensure at least 128-bit security.

Figure 8 (a) shows that implementations using the square function as an activation substitute, such as Falcon [76] and LoLa [23], yield lower performance. This is primarily because training with degree-2 polynomial activations can lead to instability issues like exploding or vanishing gradients [11, 100]. Although the square function enables faster inference, it is generally only suitable for smaller, less complex neural networks.

In contrast, private neural networks that employ approximation-based activation functions, such as OPP-CNN [54] and EVA [36], achieve higher accuracy levels. Meanwhile, TFHE's performance is significantly slower compared to implementations using interpolation approximations. This discrepancy arises because neural network architectures predominantly consist of linear layers, which benefit from the SIMD capabilities of word-wise CKKS schemes. Bit-wise TFHE, lacking SIMD support and optimized linear operations, exhibits lower performance. Additionally, the Scheme Switching method (SHE [75]) achieves the highest accuracy but at slower inference times than the approximation-based approaches.

8 Recommendation and Discussion

Considering the capabilities of each FHE method, we offer recommendations tailored to the types of operations required by a given privacy-preserving application and whether parallelism can be exploited. Figure 9 summarizes these guidelines.

First, if the application involves only linear operations, word-wise FHE methods (BGV, BFV, CKKS) are the most efficient choice. Conversely, if the application requires only non-linear operations, the next consideration should be whether the application can benefit from the word-wise SIMD technique. If SIMD does not offer an advantage, a bit-wise FHE method, such as TFHE, is the most efficient option. However, if SIMD is useful, several techniques enable non-linear functionalities in a word-wise context, including direct polynomial interpolation [85], decomposition-based polynomial interpolation [52], general functional bootstrapping [41], or special encoding methods like XCMP [7]. These word-wise methods enable parallel computation to improve efficiency.

When an application demands both linear and non-linear computations (i.e., computational generality-required applications), the next question is whether the workload can benefit from SIMD parallelism. If SIMD does not offer advantages, e.g., private sorting for an array, a bit-wise FHE method, such as TFHE, remains the most practical choice. If SIMD is beneficial, the final consideration is whether exact, error-free results are necessary. If approximate results are acceptable, as is common in tasks like neural network inference, where noise tolerance is inherent, CKKS with approximation [30, 65] offers the most efficiency. If exact computation is required, the Encoding Switching [108] between beFV and FV, and the Scheme Switching [79] between word-wise and bit-wise FHE can deliver the best performance.

While our SoK focuses on pure FHE methods for non-interactive general computation, we acknowledge related approaches with different trade-offs. Hybrid FHE-MPC protocols [48–50, 87, 101] improve efficiency by using FHE for linear operations and MPC for non-linear operations. However, these hybrid approaches require frequent client-server communication during computation, sacrificing non-interactivity. In contrast, the pure FHE methods we focus on enable servers to complete the entire computation cycle independently, without requiring the client to remain online. This non-interactive property is critical for bandwidth-constrained environments.

Another research direction is optimizing rotation key management to reduce client-to-server transmission [28, 67] or accelerate linear operations [4, 116] via specical encoding. While valuable, these works are orthogonal to our focus: we systematically analyze which FHE methods achieve functional generality supporting arbitrary combinations of linear and non-linear operations non-interactively, thereby establishing the fundamental computational capabilities and limitations of pure FHE for general AI workloads.

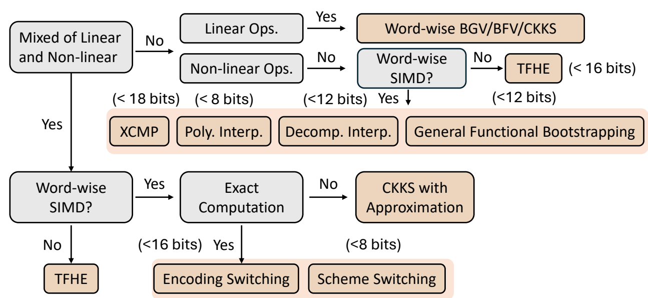

Figure 9: Flow chart to guide users to the FHE method that best fits the requirements of their privacy-sensitive applications and available resources.

9 Conclusion

In this paper, we systematically evaluate the computational generality of Fully Homomorphic Encryption (FHE)—its ability to perform non-interactive mixed linear and non-linear operations—a critical requirement for real-world privacy-sensitive applications like neural networks, graph analytics, and secure database queries. Through a three-stage methodology combining theoretical complexity analysis, micro-benchmarks on mixed-operation workloads, and empirical evaluations across five applications, we reveal significant overheads in existing general FHE solutions, emphasizing the urgent need for optimizations that bridge efficiency gaps while preserving security. To guide practitioners, we provide recommendations for selecting optimal FHE methods based on operation types, parallelism opportunities, and error tolerance, enabling developers to balance efficiency, precision, and security in privacy-preserving systems.

Acknowledgments

The authors used ChatGPT and Grammarly to check and correct any typos and grammatical errors.

References

[1] Aws aurora. https://aws.amazon.com/rds/aurora/, 2022.

[2] Microsoft azure sql server. https://azure.microsoft.com/en-us/products/azure-sql/database, 2022.

[3] Microsoft seal. https://github.com/microsoft/SEAL, 2022.

[4] Aikata Aikata and Sujoy Sinha Roy. Secure and efficient outsourced matrix multiplication with homomorphic encryption. In *International Conference on Cryptology in India*, pages 51–74. Springer, 2024.

[5] Adi Akavia, Max Leibovich, Yehezkel S Resheff, Roey Ron, Moni Shahar, and Margarita Vald. Privacy-preserving decision trees training and prediction. *ACM Transactions on Privacy and Security*, 25(3):1–30, 2022.

[6] Cuneyt Akcora, Barbara Carminati, and Elena Ferrari. Privacy in social networks: How risky is your social graph? In *2012 IEEE 28th International Conference on Data Engineering*, pages 9–19. IEEE, 2012.

[7] Rasoul Akhavan Mahdavi, Haoyan Ni, Dimitry Linkov, and Florian Kerschbaum. Level up: Private non-interactive decision tree evaluation using levelled homomorphic encryption. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 2945–2958, 2023.

[8] Ahmad Al Badawi, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, et al. Openfhe: Open-source fully homomorphic encryption library. In *proceedings of the 10th workshop on encrypted computing & applied homomorphic cryptography*, pages 53–63, 2022.

[9] Martin R Albrecht, Rachel Player, and Sam Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

[10] Andreea Alexandru, Andrey Kim, and Yuriy Polyakov. General functional bootstrapping using ckks. *Cryptology ePrint Archive*, 2024.

[11] Ramy E Ali, Jinhyun So, and A Salman Avestimehr. On polynomial approximations for privacy-preserving and verifiable relu networks. *arXiv preprint arXiv:2011.05530*, 2020.

[12] Frederik Armknecht, Colin Boyd, Christopher Carr, Kristian Gjøsteen, Angela Jäschke, Christian A Reuter, and Martin Strand. A guide to fully homomorphic encryption. *Cryptology ePrint Archive*, 2015.

[13] AWS. Enable fully homomorphic encryption with amazon sagemaker endpoints for secure, real-time inferencing, 2023. https://aws.amazon.com/cn/blogs/machine-learning/enable-fully-homomorphic-encryption-with-amazon-sagemaker-endpoints-for-secure-real-time-inferencing/.

[14] Sofiane Azogagh, Victor Delfour, Sébastien Gambs, and Marc-Olivier Killijian. Probonite: Private one-branch-only non-interactive decision tree evaluation. In *Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 23–33, 2022.

[15] Mihir Bellare, Alexandra Boldyreva, and Adam O'Neill. Deterministic and efficiently searchable encryption. In *Advances in Cryptology-CRYPTO 2007: 27th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2007. Proceedings 27*, pages 535–552. Springer, 2007.

[16] Song Bian, Haowen Pan, Jiaqi Hu, Zhou Zhang, Yunhao Fu, Jiafeng Hua, Yi Chen, Bo Zhang, Yier Jin, Jin Dong, et al. Engorgio: An arbitrary-precision unbounded-size hybrid encrypted database via quantized fully homomorphic encryption. *Cryptology ePrint Archive*, 2025.

[17] Song Bian, Zhou Zhang, Haowen Pan, Ran Mao, Zian Zhao, Yier Jin, and Zhenyu Guan. He3db: An efficient and elastic encrypted database via arithmetic-and-logic fully homomorphic encryption. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, pages 2930–2944, 2023.

[18] Alexandra Boldyreva, Nathan Chenette, Younho Lee, and Adam O'neill. Order-preserving symmetric encryption. In *Advances in Cryptology-EUROCRYPT 2009: 28th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Cologne, Germany, April 26-30, 2009. Proceedings 28*, pages 224–241. Springer, 2009.

[19] Jean-Philippe Bossuat, Rosario Cammarota, Ilaria Chillotti, Benjamin R Curtis, Wei Dai, Huijing Gong, Erin Hales, Duhyeong Kim, Bryan Kumara, Changmin Lee, et al. Security guidelines for implementing homomorphic encryption. *Cryptology ePrint Archive*, 2024.

[20] Christina Boura, Nicolas Gama, Mariya Georgieva, and Dimitar Jetchev. Chimera: Combining ring-lwe-based fully homomorphic encryption schemes. *Journal of Mathematical Cryptology*, 14(1):316–338, 2020.

[21] Zvika Brakerski. Fully homomorphic encryption without modulus switching from classical gapsvp. In *Annual cryptology conference*, pages 868–886. Springer, 2012.

[22] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6:1–36, 2014.

[23] Alon Brutzkus, Ran Gilad-Bachrach, and Oren Elisha. Low latency privacy preserving inference. In *International Conference on Machine Learning*, pages 812–821. PMLR, 2019.

[24] David Cash, Stanislaw Jarecki, Charanjit Jutla, Hugo Krawczyk, Marcel-Cătălin Roşu, and Michael Steiner. Highly-scalable searchable symmetric encryption with support for boolean queries. In *Advances in Cryptology–CRYPTO 2013: 33rd Annual Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2013. Proceedings, Part I*, pages 353–373. Springer, 2013.

[25] Gizem S Çetin, Yarkın Doröz, Berk Sunar, and Erkay Savaş. Depth optimized efficient homomorphic sorting. In *Progress in Cryptology–LATINCRYPT 2015: 4th International Conference on Cryptology and Information Security in Latin America, Guadalajara, Mexico, August 23-26, 2015, Proceedings 4*, pages 61–80. Springer, 2015.

[26] Hao Chen and Kyoohyung Han. Homomorphic lower digits removal and improved fhe bootstrapping. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 315–337. Springer, 2018.

[27] Nathan Chenette, Kevin Lewi, Stephen A Weis, and David J Wu. Practical order-revealing encryption with limited leakage. In *Fast Software Encryption: 23rd International Conference, FSE 2016, Bochum, Germany, March 20-23, 2016, Revised Selected Papers 23*, pages 474–493. Springer, 2016.

[28] Jung Hee Cheon, Minsik Kang, and Jai Hyun Park. Towards lightweight ckks: On client cost efficiency. *Cryptology ePrint Archive*, 2025.

[29] Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*, pages 409–437. Springer, 2017.

[30] Jung Hee Cheon, Dongwoo Kim, and Duhyeong Kim. Efficient homomorphic comparison methods with optimal complexity. In *Advances in Cryptology–ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part II 26*, pages 221–256. Springer, 2020.

[31] Jung Hee Cheon, Dongwoo Kim, Duhyeong Kim, Hun Hee Lee, and Keewoo Lee. Numerical method for comparison on homomorphically encrypted numbers. In *International conference on the theory and application of cryptology and information security*, pages 415–445. Springer, 2019.

[32] Jung Hee Cheon, Wootae Kim, and Jai Hyun Park. Efficient homomorphic evaluation on large intervals. *IEEE Transactions on Information Forensics and Security*, 17:2553–2568, 2022.

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

[34] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Tfhe: fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020.

[35] Reza Curtmola, Juan Garay, Seny Kamara, and Rafail Ostrovsky. Searchable symmetric encryption: improved definitions and efficient constructions. In *Proceedings of the 13th ACM conference on Computer and communications security*, pages 79–88, 2006.

[36] Roshan Dathathri, Blagovesta Kostova, Olli Saarikivi, Wei Dai, Kim Laine, and Madan Musuvathi. Eva: An encrypted vector arithmetic language and compiler for efficient homomorphic computation. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 546–561, 2020.

[37] Mark Dockendorf, Ram Dantu, and John Long. Graph algorithms over homomorphic encryption for data cooperatives. In *SECRYPT*, pages 205–214, 2022.

[38] Léo Ducas and Daniele Micciancio. Fhew: bootstrapping homomorphic encryption in less than a second. In *Annual international conference on the theory and applications of cryptographic techniques*, pages 617–640. Springer, 2015.

[39] Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive*, 2012.

[40] Robert W Floyd. Algorithm 97: shortest path. *Communications of the ACM*, 5(6):345–345, 1962.

[41] Robin Geelen and Frederik Vercauteren. Bootstrapping for bgv and bfv revisited. *Journal of Cryptology*, 36(2):12, 2023.

[42] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *International conference on machine learning*, pages 201–210. PMLR, 2016.

[43] Charles Gouert, Dimitris Mouris, and Nektarios Tsoutsos. Sok: New insights into fully homomorphic encryption libraries via standardized benchmarks. *Proceedings on privacy enhancing technologies*, 2023.

[44] Timon Hackenjos, Florian Hahn, and Florian Kerschbaum. Sagma: secure aggregation grouped by multiple attributes. In *Proceedings of the 2020 ACM SIGMOD international conference on management of data*, pages 587–601, 2020.

[45] Shai Halevi and Victor Shoup. Bootstrapping for helib. *Journal of Cryptology*, 34(1):7, 2021.

[46] Kyoohyung Han, Seungwan Hong, Jung Hee Cheon, and Daejun Park. Logistic regression on homomorphic encrypted data at scale. In *Proceedings of the AAAI conference on artificial intelligence*, volume 33, pages 9466–9471, 2019.

[47] Mingqin Han, Yilan Zhu, Qian Lou, Zimeng Zhou, Shanqing Guo, and Lei Ju. coxhe: A software-hardware co-design framework for fpga acceleration of homomorphic computation. In *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1353–1358. IEEE, 2022.

[48] Meng Hao, Hongwei Li, Hanxiao Chen, Pengzhi Xing, Guowen Xu, and Tianwei Zhang. Iron: Private inference on transformers. *Advances in neural information processing systems*, 35:15718–15731, 2022.

[49] Jiaxing He, Kang Yang, Guofeng Tang, Zhangjie Huang, Li Lin, Changzheng Wei, Ying Yan, and Wei Wang. Rhombus: Fast homomorphic matrix-vector multiplication for secure two-party inference. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pages 2490–2504, 2024.

[50] Zhicong Huang, Wen-jie Lu, Cheng Hong, and Jiansheng Ding. Cheetah: Lean and fast secure {Two-Party} deep neural network inference. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 809–826, 2022.

[51] IBM. Intesa sanpaolo and ibm secure digital transactions with fully homomorphic encryption, 2024. https://www.ibm.com/case-studies/blog/intesa-sanpaolo-ibm-secure-digital-transactions-fhe.

[52] Ilia Iliashenko and Vincent Zucca. Faster homomorphic comparison operations for bgv and bfv. *Proceedings on Privacy Enhancing Technologies*, 2021(3):246–264, 2021.

[53] Lei Jiang, Qian Lou, and Nrushad Joshi. Matcha: A fast and energy-efficient accelerator for fully homomorphic encryption over the torus. In *The Design Automation Conference (DAC 2022)*, 2022.

[54] Dongwoo Kim and Cyril Guyot. Optimized privacy-preserving cnn inference with fully homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, 18:2175–2187, 2023.

[55] Miran Kim and Kristin Lauter. Private genome analysis through homomorphic encryption. In *BMC medical informatics and decision making*, volume 15, pages 1–12. Springer, 2015.

[56] Ágnes Kiss, Masoud Naderpour, Jian Liu, N Asokan, and Thomas Schneider. Sok: Modular and efficient private decision tree evaluation. *Proceedings on Privacy Enhancing Technologies*, 2019.

[57] Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[58] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25, 2012.

[59] Mayank Kumar, Jiaqi Xue, Mengxin Zheng, and Qian Lou. Tfhe-coder: Evaluating llm-agentic fully homomorphic encryption code generation. *arXiv preprint arXiv:2503.12217*, 2025.

[60] Kristin Estella Lauter, Wei Dai, and Kim Laine. *Protecting privacy through homomorphic encryption*. Springer, 2022.

[61] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[62] Dongwon Lee, Seonhong Min, and Yongsoo Song. Functional bootstrapping for packed ciphertexts via homomorphic lut evaluation. *Cryptology ePrint Archive*, 2024.

[63] Eunsang Lee, Joon-Woo Lee, Young-Sik Kim, and Jong-Seon No. Optimization of homomorphic comparison algorithm on rns-ckks scheme. *IEEE Access*, 10:26163–26176, 2022.

[64] Eunsang Lee, Joon-Woo Lee, Junghyun Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and Woosuk Choi. Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions. In *International Conference on Machine Learning*, pages 12403–12422. PMLR, 2022.

[65] Eunsang Lee, Joon-Woo Lee, Jong-Seon No, and Young-Sik Kim. Minimax approximation of sign function by composite polynomial for homomorphic comparison. *IEEE Transactions on Dependable and Secure Computing*, 19(6):3711–3727, 2021.

[66] Joon-Woo Lee, HyungChul Kang, Yongwoo Lee, Woosuk Choi, Jieun Eom, Maxim Deryabin, Eunsang Lee, Junghyun Lee, Donghoon Yoo, Young-Sik Kim, et al. Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. *iEEE Access*, 10:30039–30054, 2022.

[67] Joon-Woo Lee, Eunsang Lee, Young-Sik Kim, and Jong-Seon No. Rotation key reduction for client-server systems of deep neural network on fully homomorphic encryption. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 36–68. Springer, 2023.

[68] Junghyun Lee, Eunsang Lee, Joon-Woo Lee, Yongjune Kim, Young-Sik Kim, and Jong-Seon No. Precise approximation of convolutional neural networks for homomorphically encrypted data. *IEEE Access*, 11:62062–62076, 2023.

[69] Seewoo Lee, Garam Lee, Jung Woo Kim, Junbum Shin, and Mun-Kyu Lee. Hetal: efficient privacy-preserving transfer learning with homomorphic encryption. In *International Conference on Machine Learning*, pages 19010–19035. PMLR, 2023.

[70] Yang Li. Checking chordality on homomorphically encrypted graphs. *arXiv preprint arXiv:2106.13560*, 2021.

[71] Zeyu Liu and Yunhao Wang. Amortized functional bootstrapping in less than 7 ms, with o˜(1) polynomial multiplications. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 101–132. Springer, 2023.

[72] Junzhen Lou. Homomorphic encryption for healthcare data privacy in industry use cases. 2024.

[73] Qian Lou, Bo Feng, Geoffrey C Fox, and Lei Jiang. Glyph: Fast and accurately training deep neural networks on encrypted data. *NeurIPS 2020 (Advances in Neural Information Processing Systems)*, 2019.

[74] Qian Lou, Feng Guo, Lantao Liu, Minje Kim, and Lei Jiang. Autoq: Automated kernel-wise neural network quantization. In *International Conference on Learning Representations (ICLR) 2020*, 2019.

[75] Qian Lou and Lei Jiang. She: A fast and accurate deep neural network for encrypted data. In *Advances in Neural Information Processing Systems (NeurIPS) 2019*, pages 10035–10043, 2019.

[76] Qian Lou, Wen-jie Lu, Cheng Hong, and Lei Jiang. Falcon: Fast spectral inference on encrypted data. *Advances in Neural Information Processing Systems*, 33:2364–2374, 2020.

[77] Qian Lou, Yilin Shen, Hongxi Jin, and Lei Jiang. Safenet: A secure, accurate and fast neural network inference. 2021.

[78] Qian Lou, Bian Song, and Lei Jiang. Autoprivacy: Automated layer-wise parameter selection for secure neural network inference. *NeurIPS 2020*, 2020.

[79] Wen-jie Lu, Zhicong Huang, Cheng Hong, Yiping Ma, and Hunter Qu. Pegasus: bridging polynomial and non-polynomial evaluations in homomorphic encryption. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1057–1073. IEEE, 2021.

[80] Wen-jie Lu, Jun-Jie Zhou, and Jun Sakuma. Non-interactive and output expressive private comparison from homomorphic encryption. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 67–74, 2018.

[81] Xianrui Meng, Seny Kamara, Kobbi Nissim, and George Kollios. Grecs: Graph encryption for approximate shortest distance queries. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 504–517, 2015.

[82] Prateek Mittal, Charalampos Papamanthou, and Dawn Song. Preserving link privacy in social network based systems. *arXiv preprint arXiv:1208.6189*, 2012.

[83] Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE symposium on security and privacy (SP)*, pages 19–38. IEEE, 2017.

[84] Koki Morimura, Daisuke Maeda, and Takashi Nishide. Accelerating polynomial evaluation for integer-wise homomorphic comparison and division. *Journal of Information Processing*, 31:288–298, 2023.

[85] Harika Narumanchi, Dishant Goyal, Nitesh Emmadi, and Praveen Gauravaram. Performance analysis of sorting of fhe data: integer-wise comparison vs bit-wise comparison. In *2017 IEEE 31st International Conference on Advanced Information Networking and Applications (AINA)*, pages 902–908. IEEE, 2017.

[86] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International conference on the theory and applications of cryptographic techniques*, pages 223–238. Springer, 1999.

[87] Qi Pang, Jinhao Zhu, Helen Möllering, Wenting Zheng, and Thomas Schneider. Bolt: Privacy-preserving, accurate and efficient inference for transformers. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 4753–4771. IEEE, 2024.

[88] Antonis Papadimitriou, Ranjita Bhagwan, Nishanth Chandran, Ramachandran Ramjee, Andreas Haeberlen, Harmeet Singh, Abhishek Modi, and Saikrishna Badrinarayanan. Big data analytics over encrypted datasets with seabed. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, pages 587–602, 2016.

[89] Raluca Ada Popa, Catherine MS Redfield, Nickolai Zeldovich, and Hari Balakrishnan. Cryptdb: Protecting confidentiality with encrypted query processing. In *Proceedings of the twenty-third ACM symposium on operating systems principles*, pages 85–100, 2011.

[90] Jean Louis Raisaro, Gwangbae Choi, Sylvain Pradervand, Raphael Colsenet, Nathalie Jacquemont, Nicolas Rosat, Vincent Mooser, and Jean-Pierre Hubaux. Protecting privacy and security of genomic data in i2b2 with homomorphic encryption and differential privacy. *IEEE/ACM transactions on computational biology and bioinformatics*, 15(5):1413–1426, 2018.

[91] Xuanle Ren, Le Su, Zhen Gu, Sheng Wang, Feifei Li, Yuan Xie, Song Bian, Chao Li, and Fan Zhang. Heda: multi-attribute unbounded aggregation over homomorphically encrypted database. *Proceedings of the VLDB Endowment*, 16(4):601–614, 2022.

[92] Ronald L Rivest, Adi Shamir, and Leonard Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[93] Nigel P Smart and Frederik Vercauteren. Fully homomorphic simd operations. *Designs, codes and cryptography*, 71:57–81, 2014.

[94] Yan-Yan Song and LU Ying. Decision tree methods: applications for classification and prediction. *Shanghai archives of psychiatry*, 27(2):130, 2015.

[95] Wenting Zheng Srinivasan, PMRL Akshayaram, and Popa Raluca Ada. Delphi: A cryptographic inference service for neural networks. In *Proc. 29th USENIX secur. symp*, volume 3, 2019.

[96] Benjamin Hong Meng Tan, Hyung Tae Lee, Huaxiong Wang, Shuqin Ren, and Khin Mi Mi Aung. Efficient private comparison queries over encrypted databases using fully homomorphic encryption with finite fields. *IEEE Transactions on Dependable and Secure Computing*, 18(6):2861–2874, 2020.
[97] Stephen Lyle Tu, M Frans Kaashoek, Samuel R Madden, and Nickolai Zeldovich. Processing analytical queries over encrypted data. 2013.
[98] Alexander Viand, Patrick Jattke, and Anwar Hithnawi. Sok: Fully homomorphic encryption compilers. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1092–1108. IEEE, 2021.
[99] Pengtao Xie and Eric Xing. Cryptgraph: Privacy preserving graph analytics on encrypted graph. *arXiv preprint arXiv:1409.5021*, 2014.
[100] Jiaqi Xue, Lei Xu, Lin Chen, Weidong Shi, Kaidi Xu, and Qian Lou. Audit and improve robustness of private neural networks on encrypted data. *arXiv preprint arXiv:2209.09996*, 2022.
[101] Jiaqi Xue, Yancheng Zhang, Yanshan Wang, Xueqiang Wang, Hao Zheng, and Qian Lou. Cryptotrain: Fast secure training on encrypted dataset. In *Proceedings of the 1st ACM Workshop on Large AI Systems and Models with Privacy and Safety Analysis*, pages 97–104, 2023.
[102] Jiaqi Xue, Yancheng Zhang, Yanshan Wang, Xueqiang Wang, Hao Zheng, and Qian Lou. Cryptotrain: Fast secure training on encrypted dataset. In *Proceedings of the 1st ACM Workshop on Large AI Systems and Models with Privacy and Safety Analysis*, pages 97–104, 2024.
[103] Zama. Build Apps with Fully Homomorphic Encryption (FHE), 2022. https://www.zama.ai/.
[104] Zama. Concrete ML: a privacy-preserving machine learning library using fully homomorphic encryption for data scientists, 2022. https://github.com/zama-ai/concrete-ml.
[105] Zama. Concrete: TFHE Compiler that converts python programs into FHE equivalent, 2022. https://github.com/zama-ai/concrete.
[106] Zama. Health insurance portability and accountability act, 2022. https://www.hhs.gov/hipaa/index.html.
[107] Zama. TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data, 2022. https://github.com/zama-ai/tfhe-rs.
[108] Yancheng Zhang, Xun Chen, and Qian Lou. Hebridge: Connecting arithmetic and logic operations in fv-style he schemes. In *Proceedings of the 12th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, pages 23–35, 2023.
[109] Yancheng Zhang, Jiaqi Xue, Mengxin Zheng, Mimi Xie, Mingzhe Zhang, Lei Jiang, and Qian Lou. Cipherprune: Efficient and scalable private transformer inference. *arXiv preprint arXiv:2502.16782*, 2025.
[110] Yuchen Zhang, Wenrui Dai, Xiaoqian Jiang, Hongkai Xiong, and Shuang Wang. Foresee: Fully outsourced secure genome study based on homomorphic encryption. In *BMC medical informatics and decision making*, volume 15, pages 1–11. Springer, 2015.
[111] Zhou Zhang, Song Bian, Zian Zhao, Ran Mao, Haoyi Zhou, Jiafeng Hua, Yier Jin, and Zhenyu Guan. Arcedb: An arbitrary-precision encrypted database via (amortized) modular homomorphic encryption. *Cryptology ePrint Archive*, 2024.
[112] Mengxin Zheng, Fan Chen, Lei Jiang, and Qian Lou. Priml: An electro-optical accelerator for private machine learning on encrypted data. In *2023 24th International Symposium on Quality Electronic Design (ISQED)*, pages 1–7. IEEE, 2023.
[113] Mengxin Zheng, Cheng Chu, Qian Lou, Nathan Youngblood, Mo Li, Sajjad Moazeni, and Lei Jiang. Ofhe: An electro-optical accelerator for discretized tfhe. In *Proceedings of the 29th ACM/IEEE International Symposium on Low Power Electronics and Design*, pages 1–6, 2024.
[114] Mengxin Zheng, Qian Lou, Fan Chen, Lei Jiang, and Yongxin Zhu. Cryptolight: An electro-optical accelerator for fully homomorphic encryption. In *Proceedings of the 17th ACM International Symposium on Nanoscale Architectures*, pages 1–2, 2022.
[115] Mengxin Zheng, Qian Lou, and Lei Jiang. Primer: Fast private transformer inference on encrypted data. *DAC 2023*, 2023.
[116] Xiaopeng Zheng, Hongbo Li, and Dingkang Wang. A new framework for fast homomorphic matrix multiplication. *Cryptology ePrint Archive*, 2023.

A Un-General FHE methods

A.1 CKKS with Polynomial Approximation

CKKS is a word-wise FHE method capable of encrypting real numbers. Since it supports approximate computation on floating-point numbers, the most common method for performing non-linear functions is polynomial approximation [66, 68]. Specifically, these methods use compositions of minimax approximation polynomials to approximate the sign function over a small interval, typically $[-1, -\epsilon] \cup [\epsilon, 1]$, with errors. Given an n-degree approximation polynomial, the complexity is $O(\sqrt{n})$ and the multiplicative depth is $\log_2 n$. The larger the interval and the smaller the error, the higher the polynomial degree required—thus increasing the number of multiplications and multiplicative depth. Most importantly, approximation can never be accurate around 0. These errors limit the applicability in domains where accuracy is critical, such as genomics [55, 90, 110] and finance [12, 46].

Overall, polynomial approximation strikes a balance between computational efficiency and accuracy, making it well-suited for applications such as neural network inference, where approximate computations are acceptable.

A.2 BGV/BFV with Polynomial Interpolation

In contrast to CKKS, computations in BFV/BGV are exact. Accordingly, non-linear functions can be implemented in BFV/BGV without approximation error. Narumanchi et al. [85] proposed computing non-linear functions via polynomial interpolation.

Take the non-linear comparison function as an example, since the comparison is a fundamental operation for many logic functions and is key to evaluating various non-linear operations [108]. The core idea is to convert the comparison between two encrypted integers, a and b, into a comparison between their difference, $z = b - a$, and zero. This is achieved by constructing a polynomial $P(x)$ with the property:

$$P(x) = \begin{cases} 1, & \text{if } x < 0; \\ 0, & \text{if } x \geq 0; \end{cases} \tag{6}$$

$P(x)$ are expressed using Lagrange interpolation:

$$P(x) = \sum_{i=0}^{p} \left(\prod_{j=0, j\neq i}^{p} \frac{x - x_j}{x_i - x_j} \right) \cdot y_i \mod p, \tag{7}$$

where x_i and y_i are chosen such that $y_i = 1$ for $x_i < 0$ and $y_i = 0$ for $x_i \geq 0$. Once the polynomial is constructed, it is evaluated homomorphically on the encrypted difference $\mathrm{Enc}(a - b)$:

$$\mathrm{Enc}(z) = P(\mathrm{Enc}(a - b)). \tag{8}$$

Upon decryption, $\mathrm{Dec}(\mathrm{Enc}(z)) = 1$ indicates $a < b$, while $\mathrm{Dec}(\mathrm{Enc}(z)) = 0$ indicates $a \geq b$.

The polynomial interpolation has notable limitations. Specifically, the standard BGV and BFV support efficient linear operations over $\mathbb{Z}_{p^r}$ [26, 45]. However, polynomial interpolation in $\mathbb{Z}_{p^r}$ is impractical, as it leads to polynomials of degree $O(p^r)$, which grow exponentially with the input bit-width. Evaluating such high-degree polynomials is prohibitive in practice [52, 96]. Therefore, non-linear operations are typically performed in the base field $\mathbb{Z}_p$. The degree of interpolation polynomials equals p, requiring $O(\sqrt{p})$ non-scalar multiplications and $O(\log_2 p)$ multiplicative depth using the Paterson-Stockmeyer algorithm. To enable efficient evaluation, the plaintext space is typically confined to a small prime p, making it unsuitable for large inputs. The value of p ranges from $2 \sim 7$ in [96], is at most 173 in [52] and at most 257 in [84]. In conclusion, this limitation makes it impractical for real-world AI applications which require general computation.

A.3 Polynomial Interpolation with Special Encoding

Aiming to enhance the capability of polynomial interpolation to support non-linear operations on large inputs, several special encoding-based methods have been proposed [52, 96], which encode a large integer as a vector of base-p digits: $a = (a_0, a_1, \ldots, a_{r-1})$, where each $a_i \in \mathbb{Z}_p$. Consequently, a non-linear operation, e.g., a comparison between two integers a and b, is transformed into a lexicographical comparison of their respective digit vectors. Formally, the process begins with the decomposition of the input integers a and b into their digit representations:

$$\begin{aligned}\mathrm{Enc}(a) &= (\mathrm{Enc}(a_0), \mathrm{Enc}(a_1), \ldots, \mathrm{Enc}(a_{r-1})),\\ \mathrm{Enc}(b) &= (\mathrm{Enc}(b_0), \mathrm{Enc}(b_1), \ldots, \mathrm{Enc}(b_{r-1})).\end{aligned} \tag{9}$$

For each digit pair a_i and b_i, a polynomial $P(\cdot)$ is constructed to evaluate the comparison $a_i < b_i$ as Equation 6. This polynomial is then applied homomorphically:

$$\mathrm{Enc}(z_i) = P(\mathrm{Enc}(a_i - b_i)). \tag{10}$$

The overall comparison is obtained by combining these digit-wise comparisons lexicographically:

$$\mathrm{Enc}(z) = \mathrm{Enc}(z_0) + \sum_{i=1}^{r-1} \mathrm{Enc}(z_i) \prod_{j=0}^{i-1} (1 - \mathrm{Enc}(z_j)), \tag{11}$$

where $\prod_{j=0}^{i-1}(1 - \mathrm{Enc}(z_j))$ ensures that less significant digits only contribute when more significant digits are equal.

By decomposing the large field into smaller subfields and performing comparisons at the digit level, this method achieves a significantly lower circuit depth. The depth of the circuit evaluating the comparison of two b-bit integers is given by:

$$\log_2 \log_p 2^b + \log_2(p-1) + 4. \tag{12}$$

In contrast, directly interpolation on $\mathbb{Z}_{p^r}$ has a multiplicative depth of $O(\log_2 p^r)$. This reduced depth makes the method more practical for larger fields. By addressing the challenge of deep circuit depth in polynomial interpolation, [52] enables faster homomorphic non-linear operations for practical applications in BGV/BFV. Despite its significant efficiency improvement, the special encoding-based ciphertexts do not support linear operations since the number is expressed in the vector form, making it not an FHE method enabling general computation.

Similarly, other special encoding-based approaches have also been proposed to enhance the efficiency of non-linear operations in word-wise FHE. Unfortunately, these methods sacrifice generality, as the specialized ciphertext formats do not support linear operations. Below, we present a representative example.

A.4 Exponential Encoding (XCMP)

The exponential encoding method for FHE private comparison, named XCMP, was first proposed by Lu et al. [80]. The core idea is based on the fact that the message space in HE is a polynomial ring, i.e., $\mathbb{Z}_p[x]/\langle x^n + 1\rangle$, and that ciphertext multiplication corresponds to polynomial multiplication. This enables the design of a special encoding scheme that encodes the values to be compared into the polynomial's degree coefficients.

The comparison process of integers a and b begins with encoding a and $-b$ as polynomials X^a and X^{-b}, respectively. These polynomials are encrypted into ciphertexts $\mathrm{Enc}(X^a)$ and $\mathrm{Enc}(X^{-b})$, which are used to perform:

$$\mathrm{Enc}(C(X)) = T(X) \times \mathrm{Enc}(X^a) \times \mathrm{Enc}(X^{-b}) \mod (X^n + 1) \tag{13}$$

where $T(X) = 1 + X + \cdots + X^{n-1}$. Note that the polynomial with a negative degree $X^{-b} \equiv -X^{n-b} \mod (X^n + 1)$.

Finally, decrypt $\mathrm{Enc}(C(X))$ and evaluates the 0-th coefficient $C(X)[0]$, which indicates the comparison result:

- If $a \le b$, X^{b-a} from $T(X)$ aligns with X^{a-b}, yielding $C(X)[0] = 1$.
- If $a > b$, the $(n - (a - b))$-th term of $T(X)$ aligns with X^{a-b}, resulting in $C(X)[0] = -1$ due to the wrap-around: $X^{n-(a-b)} \times X^{a-b} \equiv -1 \mod (X^n + 1)$.

In other words, $C(X)[0] = 1$ if $a \le b$, and $C(X)[0] = -1$ if $a > b$.

XCMP has a multiplicative depth and complexity of 1 due to its use of a single ciphertext multiplication. However, it is limited by a small input field, requiring both a and b to be smaller than the polynomial degree n. This constraint reduces efficiency for large inputs, as a larger n is needed [80]. Although there have been discussions about extending the domain to $\mathbb{F}_{n^2}$ by decomposing numbers in base n and processing digits separately:

$$\mathbb{I}[a \le b] = \mathbb{I}[a_1 \le b_1] + \mathbb{I}[a_1 = b_1] \cdot \mathbb{I}[a_0 \le b_0], \tag{14}$$

where the equality check result $\mathbb{I}[a_1 = b_1]$ must be converted to the XCMP format for multiplication with $\mathbb{I}[a_0 \le b_0]$. This conversion involves Fermat's little theorem, resulting in a large multiplicative depth. As the result, XCMP performs well for small domains (less than 16 bits). For larger domains, performance significantly decreases [80].

A.5 General Functional Bootstrapping

Functional bootstrapping [10, 62, 71] extends traditional bootstrapping techniques by not only refreshing ciphertexts but also by enabling the homomorphic evaluation of arbitrary functions on encrypted data. The key idea is to express a target function as a look-up table (LUT) and then approximate this LUT via a constructed interpolation polynomial. Lee et al. [62] proposed functional bootstrapping for BFV, building upon regular BFV-style bootstrapping [41]. However, BFV-style bootstrapping efficiently supports only a small number of slots. To improve efficiency, Alexandru et al. [10] employed CKKS-style bootstrapping to increase slot utilization. Specifically, the target function is first encoded as a LUT over a finite base-p field $\mathbb{Z}_p$, and then approximated using a trigonometric Hermite interpolation polynomial $R(x)$. For instance, in the first-order case, the interpolation is constructed so that:

$$R\left(\frac{k}{p}\right) = f(k) \text{ for all } k \in \{0, 1, \ldots, p-1\} \text{ and } R'\left(\frac{k}{p}\right) = 0 \tag{15}$$

This polynomial is then evaluated homomorphically using an efficient technique such as the Paterson–Stockmeyer algorithm. The complexity of evaluating a polynomial of degree d using this method is roughly proportional to $\sqrt{2d} + \log d$. For a first-order interpolation, where $d = p - 1$, the complexity increases approximately as $\sqrt{p}$. Moreover, if further noise reduction is required, higher-order interpolations can be used (e.g., second-order or third-order), which increase the polynomial degree (to about $\frac{3p}{2}$ or $2p - 1$, respectively) and thus the overall cost. Moreover, extending the input space from $\mathbb{Z}_p$ to $\mathbb{Z}_{p^r}$ requires representing each integer as a vector of base-p digits, similar to special encoding-based polynomial interpolation in BFV/BGV [52, 108], which precludes the continuous evaluation of linear and non-linear functions on large inputs. Experiments in [10] demonstrate that general functional bootstrapping remains practical only for LUT evaluations up to 12-bit inputs.