An Adaptive User-Guided Resilient Approach for Dynamic Bootstrapping in Homomorphic Encryption

S. Yasmin
Department of Computer Applications, B.S. Abdur Rahman Crescent Institute of Science & Technology, Chennai, India
yasmins_ca_jan21@crescent.education

G. Shree Devi
Department of Computer Applications, B.S. Abdur Rahman Crescent Institute of Science & Technology, Chennai, India
shreedevi@crescent.education (corresponding author)

Received: 10 May 2025 | Revised: 29 May 2025 and 11 June 2025 | Accepted: 16 June 2025

 DOI: https://doi.org/10.48084/etasr.12045

ABSTRACT

Cheon-Kim-Kim-Song (CKKS)-based Homomorphic Encryption (HE) allows encrypted data to undergo approximate calculations. This makes it particularly suitable for real-world applications that rely on floating-point operations, like signal processing and encrypted machine learning. Despite this advantage, most current systems use fixed bootstrapping schedules that activate regardless of the actual noise level in the encrypted data. This inflexible design leads to unnecessary bootstrapping, higher memory usage, and slower processing, especially when dealing with different data formats and file sizes. To overcome these challenges, we introduce the Adaptive User-guided Resilient Approach using CKKS (AURA-CKKS), a new encryption method featuring a dynamic, noise-sensitive bootstrapping process. In order to help the system decide whether bootstrapping is required, the AURA-CKKS algorithm first accepts user-defined parameters, such as noise thresholds and bootstrapping preferences. Before calculations start, the algorithm estimates noise growth by doing an initial examination of the ciphertext parameters. The algorithm constantly checks noise levels during encrypted operations to ensure that bootstrapping is only activated when required, improving efficiency and preventing unnecessary calculations. Throughout the homomorphic operation cycle, this adaptive technique preserves the integrity of the ciphertext, minimizes processing time, and permits effective management of computational resources. Test results show that AURA-CKKS can boost bootstrapping efficiency by up to 46%, reduce memory usage by around 39%, and increase processing speed by over 51% compared to standard CKKS methods. This positions AURA-CKKS as a powerful and adaptable solution for secure, encrypted computation. Experimental results demonstrate that AURA-CKKS significantly outperforms existing CKKS implementations in terms of throughput, scalability, and noise management, making it a practical and efficient solution for secure computation.

Keywords-homomorphic encryption; CKKS, AURA-CKKS; adaptive bootstrapping; noise threshold; encrypted computation; memory optimization; throughput enhancement; selective encryption

I. INTRODUCTION

Advances in encryption, particularly those related to cloud computing and AI-based data analysis, are driving a growing interest in privacy-preserving technologies like Fully Homomorphic Encryption (FHE), Secure Multiparty Computation (SMPC), and Zero-Knowledge Proofs (ZKPs). Post-Quantum Cryptography (PQC), especially lattice-based schemes like such as NTRU and Ring-LWE, which also serve as the foundation for many Homomorphic Encryption (HE) techniques, has accelerated due to the development of quantum computing. With HE at their core, these developing paradigms seek to offer effective, scalable, and secure solutions for safeguarding private information in cooperative and outsourced settings.

HE is ideal for secure outsourcing, encrypted machine learning, and privacy-preserving analytics since it enables computation on encrypted data without the need for decryption. An important feature of the Cheon–Kim–Kim–Song (CKKS) scheme is its ability to enable approximate arithmetic over real and floating-point integers, which is necessary for applications

such as neural network inference, statistical modeling, and signal processing.

Notwithstanding its benefits, CKKS has significant practical drawbacks. Every homomorphic process adds noise to the ciphertext, making decryption difficult above a certain threshold [1]. Conventional CKKS uses bootstrapping at predetermined intervals, which results in overhead and redundant procedures [2]. Its rigid workflows lack adaptability to data features like size or type [3]. Most implementations precompute all evaluation keys (e.g., rotation, relinearization, bootstrapping), consuming memory even if they are not needed [4].

Recent studies have focused on optimizing CKKS bootstrapping to improve accuracy and performance. Among the noteworthy developments are methods designed for binary ciphertexts [5], methods that use polynomial approximations and inverse sine functions to increase numerical precision [6], and improved full-RNS implementations that achieve higher accuracy and efficiency without the need for sparse secret keys [1]. These methods still use static or preset refresh mechanisms even though they enhance certain elements of bootstrapping.

The adaptive, noise-threshold-aware bootstrapping method introduced by Adaptive User-guided Resilient Approach using CKKS (AURA-CKKS) only activates when the noise budget of the ciphertext drops below a threshold that the user specifies. This strategy avoids needless bootstrapping, saves processing resources, and extends the ciphertext utility during encrypted evaluation, in contrast to static or interval-based alternatives. Unlike conventional static bootstrapping, it enhances scalability and efficiency, as indicated in Table I.

In addition to adaptive bootstrapping, AURA-CKKS presents several other innovations:

- User-guided policies: Allows file-level encryption parameter customization for different data types (e.g., text, audio, and image).
- Context-aware strategy: Automatically adjusts encryption and bootstrapping based on file size, structure, and type.
- On-demand key generation: Generates evaluation keys only when needed, reducing memory overhead.
- Real-time monitoring: Tracks noise budget, bootstrapping frequency, and throughput for dynamic optimization.
- Improved scalability**:** Delivers better speed, memory usage, and throughput across diverse workloads than CKKS, Brakerski/Fan-Vercauteren (BFV), and Brakerski–Gentry–Vaikuntanathan (BGV).

TABLE I. COMPARATIVE ANALYSIS OF BOOTSTRAPPING TECHNIQUES IN CKKS-BASED FRAMEWORKS

Dimension	**AURA-CKKS**	**[1]**	**[7]**	**[8]**	**[6]**	**[5]**
Goal	Adaptive, file-aware encryption with noise threshold-based dynamic bootstrapping	Boost CKKS bootstrapping's accuracy and effectiveness in a full-RNS environment	Generalize full-RNS CKKS for reduced temporary moduli and efficient sine approximation	Evaluate various bootstrapping methods and their performance in open-source libraries	Improve bootstrapping with high precision by employing minimax polynomials	For ciphertexts encoding binary values, enable bootstrapping
Target data type	Actual files using CKKS vector encoding (text, images, etc.)	Vectors of general numbers (CKKS)	Vectors of real numbers (CKKS)	Vectors of real numbers (CKKS)	Value approximation (CKKS)	Binary (bit-level) data in CKKS
Bootstrapping trigger	User-defined noise threshold with real-time monitoring	Calculation at fixed depth with integrated bootstrapping	Fixed-depth computation with integrated bootstrapping	Analysis of existing methods without introducing new triggers	Consistent bootstrapping in any situation	Only ciphertexts containing binary-encoded values are subject to bootstrapping
Adaptivity	Dynamic bootstrapping when noise threshold is crossed	Static — built-in circuit depth bootstrapping	Static bootstrapping based on circuit depth	Comparative analysis without introducing new adaptivity mechanisms	Static — uses heavy function approximation	Static — predefines the use case for binary
Throughput optimization	Prevents unnecessary bootstrapping	Absence of a clear performance emphasis	Reduces number of non-scalar multiplications by about half compared to Chebyshev method	Evaluates throughput across different schemes	Emphasizes precision over speed	Not tuned for performance

II. RELATED WORK

By allowing processing on encrypted data without decryption, HE provides privacy benefits over more conventional techniques like Rivest–Shamir–Adleman (RSA), ElGamal, and Paillier [9]. Although there are still issues with high latency, ciphertext extension, and parameter tweaking, FHE approaches, including multi-key FHE, BGV, and Dijk–Gentry–Halevi–Vaikuntanathan (DGHV), have been investigated for cloud security [10-13]. Lattice-based cryptography has also been explored for HE integration due to its low decryption complexity [11].

Researchers have suggested HE-based General Matrix Multiplication (HEGMM) using Single Instruction Multiple Data (SIMD) for faster encrypted matrix processing [14] to increase performance, and Partially Homomorphic Encryption (PHE) for secure distributed estimation with reduced overhead

[15, 16] and. Integrating Trusted Execution Environments (TEEs) into FHE, particularly through Paillier-based Partially FHE (PFHE), improves secure multiplication and trust [17]. Applications that protect privacy, such as secure medical image diagnosis, are made possible by HE in conjunction with deep learning and blockchain [18]. Error handling is a major distinction between FHE and Somewhat Homomorphic Encryption (SWHE). While SWHE is constrained by noise growth, FHE employs bootstrapping to manage noise and enable infinite operations [19]. To solve data ownership problems in shared computing environments, a Multi-Key FHE (MK-FHE) architecture with a verifiable Homomorphic Message Authenticator (HMAC) has been developed. This method maintains computational integrity while enabling precise statistics [20].

III. METHODOLOGY

The proposed methodology utilizes AURA-CKKS to enable efficient and adaptive HE across diverse data types and workloads. By including a user-configurable, runtime-adaptive layer that dynamically modifies encryption settings according to input characteristics and user-defined constraints like security, latency, and precision, it improves on the conventional CKKS approach. The data flow across the discrete, modular components of the AURA-CKKS system is depicted in Figure 1.

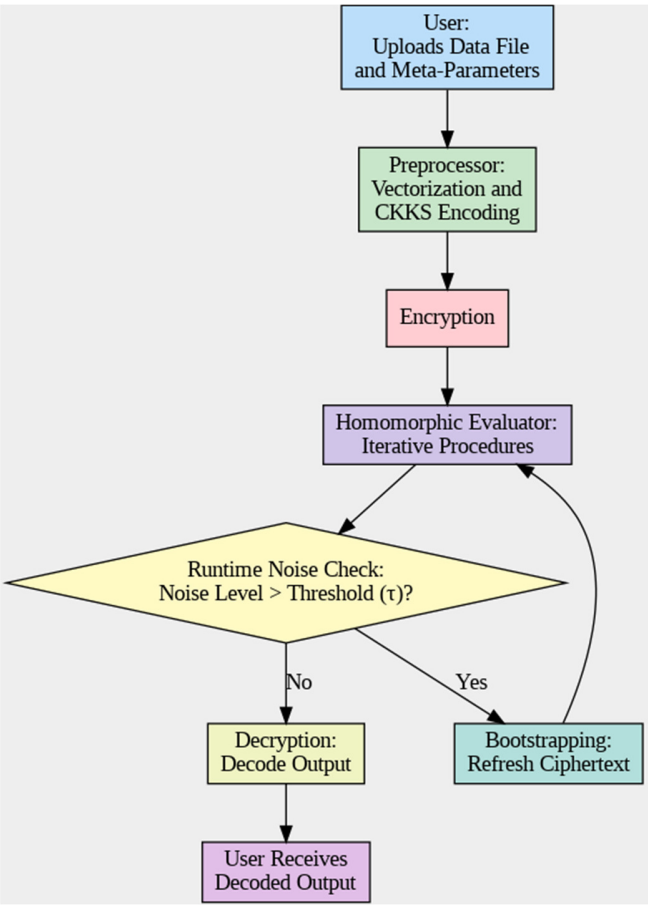

Fig. 1. AURA-CKKS system data flow.

Using the Frontend GUI, the user uploads a data file and meta-parameters, which are then sent to the Preprocessor for vectorization and CKKS encoding. After being encrypted, the data are sent to the Homomorphic Evaluator, where iterative procedures are performed. A runtime noise check is carried out during this calculation phase, and bootstrapping is initiated to refresh the ciphertext if the noise level exceeds the threshold (τ). The user receives the decoded output at the end of the Decryption step. This modular flow guarantees scalability, noise reduction, and effective encrypted computing.

Input processing, metadata interpretation, encoding, parameter selection, encryption, optional bootstrapping, encrypted computation, and decryption are all steps in the structured pipeline that AURA uses. Using heuristics or user-defined metadata/UI preferences, AURA incorporates customizable intelligence at every stage. Following data encoding into complex vectors, AURA adjusts the CKKS parameters (ring dimension, modulus depth, and scaling factor) according to priorities and workload. Bootstrapping is used if noise levels approach the predefined threshold. To ensure safe and effective processing, all computations are performed on encrypted data, and decryption and decoding occur only at the very end.

This approach not only preserves the core advantages of CKKS (approximate arithmetic on encrypted vectors) but also enhances usability, performance, and applicability across varied computational scenarios by tuning the cryptographic behavior to the workload's depth, data type, and performance constraints.

A. Overview of AURA-CKKS Processing Phases

1) User Input Phase

In the user input phase, the AURA-CKKS system begins by accepting two primary types of input from the user:

- Data File (D): Users primarily wish to encrypt and process these data in a homomorphic manner. Text, pictures, audio, video, are among the supported formats. By extracting pertinent characteristics or numerical representations, the system transforms these into intricate vector formats that are compatible with CKKS.
- Choosing a configuration file or GUI: In addition to the data, the user provides meta-parameters that guide AURA-CKKS's handling of the data via a configuration file or GUI. These meta-parameters form the tuple shown in (1):

$$\text{Meta} = \{\text{data\_type}, \text{precision\_level}, \text{latency\_pref}, \text{security\_bits}\} \quad (1)$$

where:

- data_type*:* Indicates the type of input data, enabling AURA-CKKS to select the appropriate encoding and parameter scale. For instance, the term "image" is transformed into pixel matrices, "text" is numerically encoded and tokenized, "audio" is transformed into frequency-domain vectors, and "video" is transformed into time-sequenced picture frames.

- precision_level: Defines the required numerical accuracy level for the calculation. It establishes the modulus chain depth indirectly and the CKKS scaling factor (Δ). High precision (≈240) yields more accurate results but requires longer chains and deeper levels. Low precision (≈220) is faster but less accurate yet. This relationship is mathematically defined as:

  $\Delta = 2^{\text{precision\_bits}}$ e.g., $\Delta = 2^{40}$ for high precision

- latency_pref: This parameter indicates the user's preferred processing speed, which assists AURA balance computation time and accuracy: The "low_latency" setting selects shallower levels with less bootstrapping, the "high_accuracy" setting permits longer computations with deeper modulus chains, and the "energy_efficient" setting focuses on reducing noise growth. Internally, this modifies the number of allowable operations L before bootstrapping:

  $L_{max} = f(\text{latency\_pref})$

- security_bits: Indicates the desired level of encryption security in bits (e.g., 128, 192, 256) and defines the ring dimension N (e.g., 2^{15} or 2^{16}), the selection of primes q_i in the modulus chain, and the defense against lattice-based assaults. This is formally guaranteed by use of:

  $\lambda = \text{security\_bits} \Rightarrow N, q_i \text{ satisfy HE standard constrains}$

2) Preprocessing and Encryption Phase

AURA-CKKS system converts real-world data into ciphertext appropriate for homomorphic computing by processing the input data file (D) through the preprocessing and encryption stages. This procedure entails:

- Data vectorization (preprocessing phase): First, the raw input file D, which may be text, audio, or an image, is transformed into a real-valued vector, as defined in (2):

  $$x = [x_1, x_2, \dots, x_n] \in \mathbb{R}^n \quad (2)$$

  where each element x_i denotes pixel intensity if D is an image. If D is audio, then x_i denotes the waveform's sampled amplitudes If D is text, then x_i denotes the token embeddings. By using vectorization, the input data are guaranteed to be in a format that can be encoded into the CKKS plaintext space.

- CKKS encoding: The real vector is then converted into a complex-valued polynomial that is encrypted by AURA-CKKS using the CKKS encoder defined in (3):

  $$\tilde{x} = \text{Encode}(x, \Delta) \quad (3)$$

  where $\tilde{x} \in \mathbb{C}[X]/(X^N + 1)$ is the encoded plaintext polynomial and $\Delta \in \mathbb{R}^+$ is the scaling factor, selected according to the user-specified precision_level. This encoding approximately preserves mathematical fidelity while mapping the real vector into a plaintext polynomial.

- Encryption with public key: The first ciphertext is obtained by encrypting the encoded plaintext $\tilde{x}$ with the CKKS public key pk, yielding the initial ciphertext defined by (4):

  $$ct_0 = \text{Encrypt}(\tilde{x}, pk) \quad (4)$$

  where $ct_0 \in C$ is the ciphertext in the CKKS ciphertext space $C \subseteq R_q^2$, with $R_q = \mathbb{Z}_q[X]/(X^N + 1)$. The ciphertext is an approximate encryption, satisfying $ct_0 \approx x + e$, with $\|e\| < \Delta$. The condition $\|e\| < \Delta$ ensures that the noise does not overwhelm the plaintext values during homomorphic operations.

3) Operation Execution Phase (on Encrypted Data)

As soon as the encrypted ciphertext ct_0 is prepared, AURA-CKKS evaluates the intended computational function homomorphically. This function is denoted as $f(\cdot): \mathbb{R}^n \to \mathbb{R}^n$. Any arithmetic circuit that the user wants to evaluate, such as a filtering operation, a polynomial transformation, or a neural network layer, can be represented by this function. The CKKS method supports additions and multiplications, which are included in the iterative operations executed in encrypted form over T steps as shown in (5):

$$ct_i = f(ct_i - 1), \text{ for } i = 1,2, \dots, T \quad (5)$$

However, the ciphertext's noise level increases with each homomorphic multiplication. One way to model this growth is $\text{Noise}(ct_i) \approx \text{Noise}(ct_{i-1}) \cdot \Delta'/\Delta$. Following multiplication, Δ' is the scaling factor (often $\Delta' > \Delta$). The entire cumulative encryption error in the ciphertext is measured by noise (ct_i).

4) Bootstrapping Check Phase

To address the rising levels of noise, a runtime-adaptive bootstrapping is introduced by AURA-CKKS. The system checks the noise threshold following the operation defined by (6) and (7):

$$\text{If } \text{Noise}(ct_i) > \tau, \text{ then} \quad (6)$$

$$ct_i' \leftarrow \text{Bootstrap}(ct_i \quad (7)$$

where:

- τ is the predefined noise tolerance threshold, determined based on the desired decryption accuracy.

- $\text{Bootstrap}(\cdot)$ is the CKKS bootstrapping function that re-encrypts the data and reduces the noise level.

Bootstrapping performs the following functions:

- Refreshes ct_i by replacing it with a new ciphertext ct_i' that encodes the identical plaintext for ct_i.

- Resets the modulus chain and scale, which homomorphic multiplications have previously consumed.

- Maintains semantic integrity by keeping the underlying plaintext essentially unchanged:

  $ct_i' = \text{Bootstrap}(ct_i) \approx x_i + e_{new}, \text{with } \|e_{new}\| < \Delta$

5) Decryption Phase

Once all homomorphic operations (and any intermediate bootstrapping) are completed, the system holds the final ciphertext ct_T. This ciphertext is then decrypted using the secret key (denoted sk) to retrieve the approximate encoded result as in (8):

$$\tilde{y} = \text{Decrypt}(ct_T, sk) \quad (8)$$

After decryption, decoding is performed using the inverse of the CKKS encoding scheme as in (9):

$$y = \text{Decode}(\tilde{y}, \Delta') \quad (9)$$

The overall transformation can be summarized as

$$y = \text{Decode}(\text{Decrypt}(ct_T, sk), \Delta')$$

B. AURA-CKKS Workflow

Algorithm 1 demonstrates the general steps of the AURA-CKKS system: user input, preprocessing and encryption, iterative homomorphic computation with optional bootstrapping, and final decryption and output.

```
Algorithm 1: AURA-CKKS workflow for
Adaptive HE
Inputs:
D: User data file (text,image,audio)
   Meta={data_type,precision_level,latency_
   pref,security_bits}
Outputs:
y: Final decoded plaintext output
Procedure:
1. User input phase:
   (a) Upload data file D and meta-
   parameters
   (b) Extract: data_type, precision_level,
   latency_pref, security_bits
2. Preprocessing & encryption phase:
   (a) x ← Vectorize(D)
   (b) Δ ← 2^(precision_level)
   (c) x̃ ← Encode(x,Δ)
   (d) ct₀ ← Encrypt(x̃,pk)
3. Computation phase:
   ct_current ← ct₀
   for i = 1 to T do
      ct_next ← Apply_Homomorphic_Op(ct_
      current)
      if Noise(ct_next) > noise_threshold,
      then
         ct_next ← Bootstrap(ct_next)
      end if
      ct_current ← ct_next
   end for
4. Decryption phase:
   ỹ = Decrypt(ct_T,sk)
   y=Decode(ỹ,Δ')
5. Output phase:
   Return y
```

The main notations and symbols used in this paper are listed in Table II, together with brief descriptions to ensure uniformity and clarity in the mathematical formulation and pseudocode representation of the AURA-CKKS system.

C. Performance Evaluation Using Real-World Dataset

To thoroughly evaluate the performance of our novel AURA-CKKS encryption scheme, we used 50 different English files from the Project Gutenberg repository to test AURA-CKKS on text data [21]. This dataset covers a broad spectrum of syntactic structures, vocabulary complexity, and document lengths. The efficacy and efficiency of the scheme in managing the subtleties of natural language can be evaluated by testing it on this diverse corpus.

TABLE II. KEY NOTATIONS AND SYMBOLS

Symbol	Description
D	User data file (text, image, audio, video)
Meta	{data_type, precision_level, latency_pref, security_bits}
$x \in \mathbb{R}^n$	Vectorized form of input data
$\tilde{x} = \text{Encode}(x, \Delta)$	CKKS-encoded plaintext with scale Δ
Δ	Initial scale factor (based on precision_level)
pk, sk	Public and secret key pair for encryption/decryption
ct_0	Initial ciphertext: $ct_0 = \text{Encrypt}(\tilde{x}, pk)$
ct_i	Ciphertext at i-th operation: $ct_i = f(ct_i - 1)$
$f(\cdot): \mathbb{R}^n \to \mathbb{R}^n$	Homomorphic function (e.g., filter, classifier)
Noise (ct_i)	Noise level of ciphertext after i operations
Δ′	Updated scale after homomorphic multiplication
T	Noise threshold for triggering bootstrapping
ct_i'	Refreshed ciphertext after bootstrapping
$\tilde{y}$	Decrypted CKKS plaintext: $\tilde{y} = \text{Decrypt}(ct_T, sk)$
y	Final decoded output: $y = \text{Decode}(\tilde{y}, \Delta')$

1) Graphical Representation of Dataset Properties

The following bar charts illustrate how input variety affects AURA-CKKS performance by summarizing important aspects of the 50 text files.

Figures 2(a) and 2(b) illustrate the estimated memory usage and encryption throughput for the 50 text files, respectively. Figure 2(a) shows a steady memory footprint with minimal variation across all files, as depicted by the purple bars that present memory consumption per file, which remains continuously around 350 MB. The encryption throughput (B/s) for 50 text files is shown on the right side of Figure 2(b). The throughput of each file is represented by a teal bar, which is based on the size of the file and the simulated encryption duration. In contrast to the consistent memory utilization, throughput fluctuates greatly; some files have low values, whereas others reach a peak of about 500,000 B/s. This variation is consistent with the simulation, which shows that for files with shorter encryption periods exhibit higher throughput.

Figure 3 illustrates the decryption throughput for the 50 text files. The vertical axis quantifies the speed of decryption in B/s. The horizontal axis represents each of the 50 text files, numbered sequentially from 1 to 50. Each coral-collared bar corresponds to a specific file, and its height indicates the calculated decryption throughput for that file, determined by the file size and simulated decryption time.

Figure 4 shows the noise budget per file using yellow bars, illustrating the remaining noise capacity in each of the 50 encrypted text files after simulated homomorphic computations. The height of each bar, expressed in unitless

terms, represents the available noise budget: taller bars mean there is more space for additional operations without risking decryption failure, whereas shorter bars indicate the ciphertext is getting close to its noise threshold. Beyond this threshold, additional computation could jeopardize the accuracy of decryption.

Certain file indices with much lower noise budgets, including 1, 11, 19, 37, and 41, are highlighted in the graph. These indicate locations where the threshold for ciphertext noise has almost been surpassed. In such circumstances, if more homomorphic operations are expected, the AURA-CKKS system will proactively activate its adaptive bootstrapping technique to minimize noise. Thus, the graph not only shows the amount of noise that accumulates across encrypted files, but it also indicates when the intelligent, noise-aware bootstrapping of AURA-CKKS would be used to preserve computation integrity.

Figure 5 illustrates the noise budget change per file, highlighting AURA-CKKS's bootstrapping activity by showing how the noise budget fluctuates during file processing. Bootstrapping events that restore computational capability and lower ciphertext noise are represented by positive spikes. On the other hand, values that are negative or close to zero represent normal noise consumption from common homomorphic computations, suggesting a slow build-up of noise over time.

AURA-CKKS uses a noise-level-based adaptive bootstrapping approach, with bootstrapping events clearly indicated by noticeable noise budget spikes at file indices 9, 17, 29, and 43. This confirms AURA-CKKS's effective use of resources for safe homomorphic computing, showing that bootstrapping only begins when noise approaches a critical threshold.

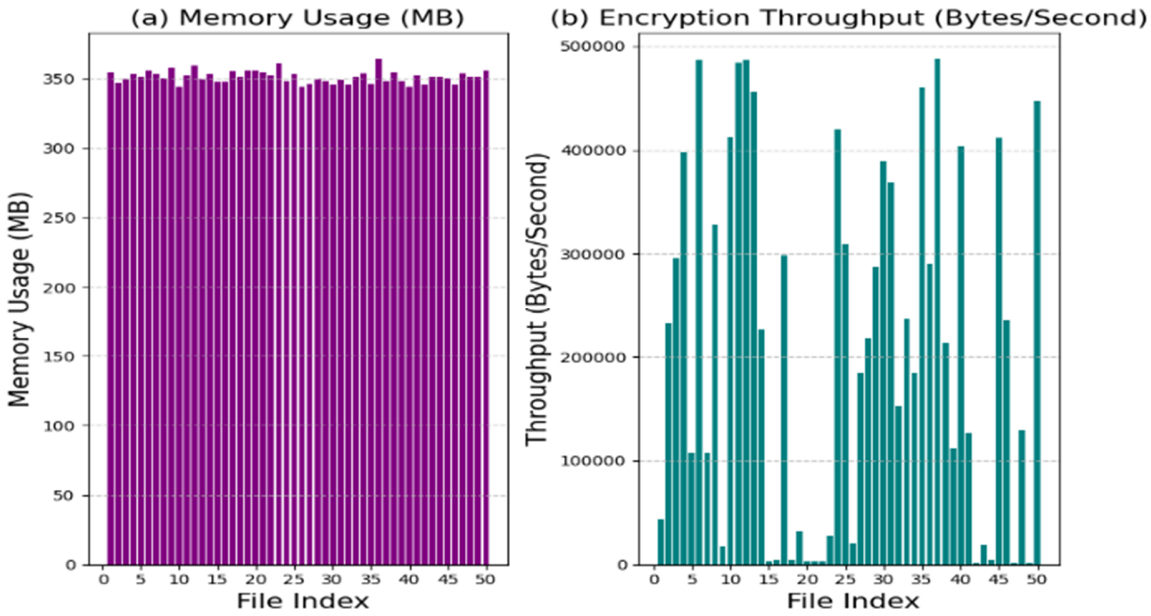

Fig. 2. (a) Memory usage and (b) encrypted throughput of AURA-CKKS across 50 text files.

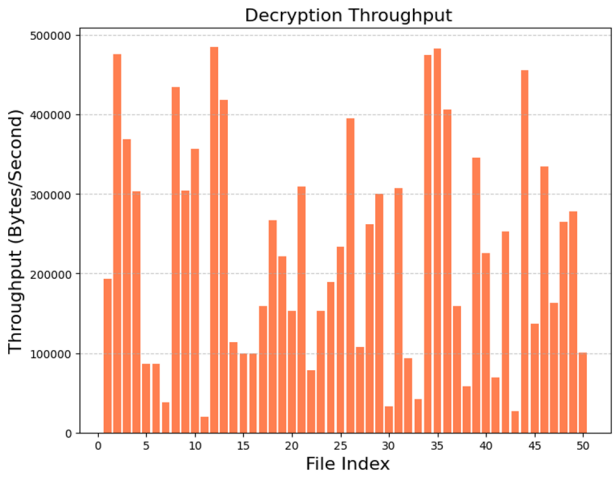

Fig. 3. Decryption throughput for 50 text files.

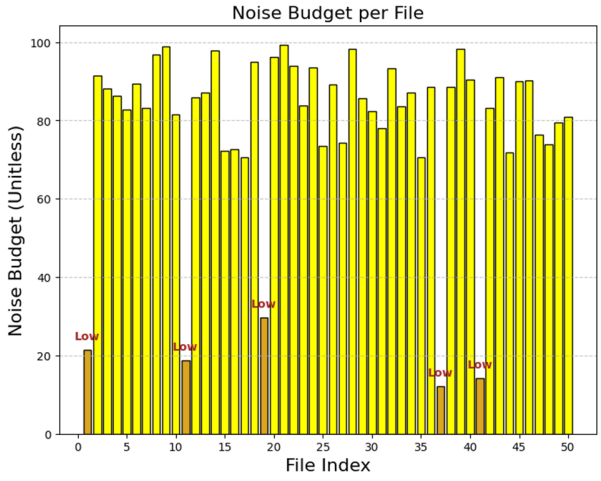

Fig. 4. Noise budget per file for 50 text files.

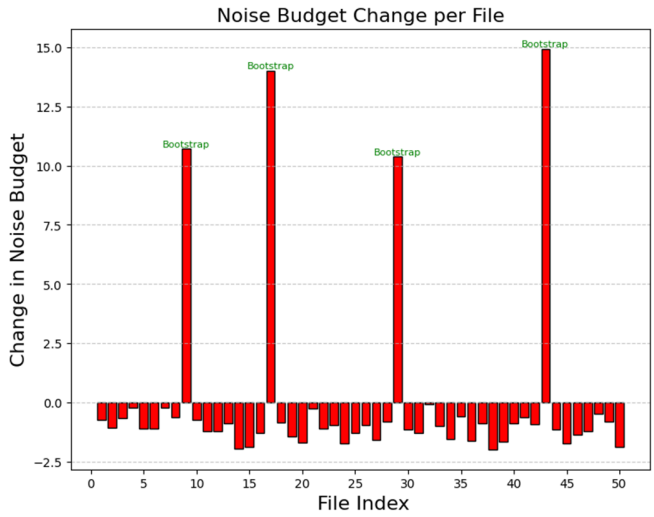

Fig. 5. Noise budget change per file for 50 text files.

IV. RESULTS

Significant performance differences are observed when comparing BFV, BGV, CKKS, and the suggested AURA-CKKS, with an emphasis on noise tolerance, memory utilization, and throughput for file sizes ranging from 100 KB to 1 MB. BFV works well in applications that call for precise discrete computations, supporting exact integer arithmetic [22]. BGV focuses on integer operations and provides greater parameter flexibility, controlling noise increase by modulus switching [23]. CKKS is effective for continuous encrypted data, such as images and signals, because it can perform approximate arithmetic on real or complex numbers [1]. These design variations highlight important trade-offs when selecting HE systems according to application requirements. AURA-CKKS, particularly with larger datasets, yields significant efficiency advantages across a variety of workloads.

Figure 6 shows a comparison of the performance of several HE methods. In Figure 6(a), BGV exhibits the lowest noise threshold before bootstrapping is required, followed by BFV, CKKS, and AURA-CKKS, which maintains the highest noise tolerance. In Figure 6(b), AURA-CKKS outperforms CKKS, BGV, and BFV, which exhibits the highest memory use. Finally, Figure 6(c) demonstrates AURA-CKKS's superior processing speed, showing that it attains the maximum throughput (MB/s). CKKS follows, whereas BFV and BGV fall behind.

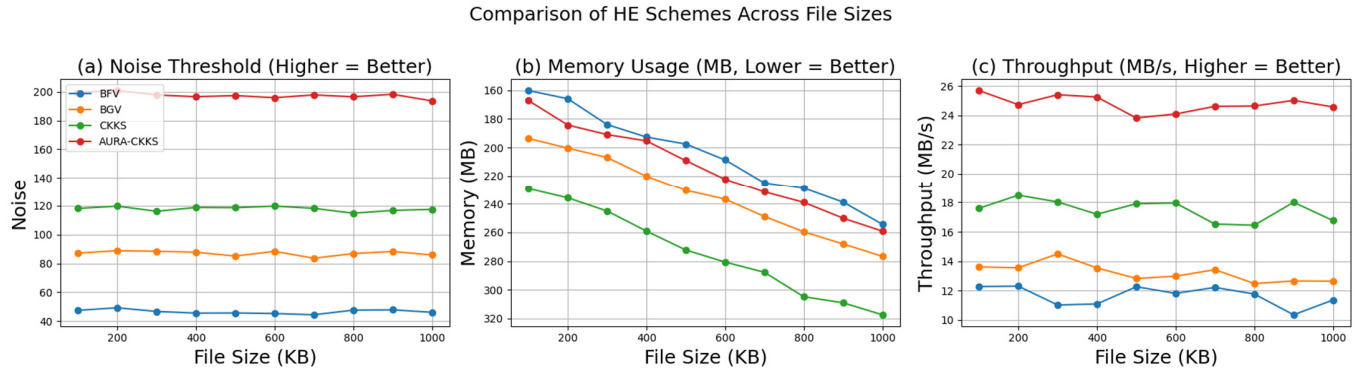

Fig. 6. Comparison of HE methods across file sizes: (a) noise threshold, (b) memory usage, and (c) throughput.

A distinct pattern emerges when examining cumulative noise growth without bootstrapping across various file sizes and types. Noise rises with file size for all formats (text, image, and audio). Text files exhibit the slowest increase in noise, whereas audio files accumulate noise the fastest, surpassing the bootstrapping threshold at smaller sizes. This demonstrates how the timing of bootstrapping and noise accumulation in HE systems are greatly influenced by data properties, including encoding and complexity.

Figure 7 illustrates the inherent challenge of noise accumulation in HE. It plots the cumulative noise level in HE ciphertexts as a function of file size for different data types (text, image, and audio) without bootstrapping:

- Exponential noise growth: In line with the intrinsic noise increase during HE operations, the graph shows a nonlinear increase in noise with file size. This suggests that larger files introduce higher noise due to increased computational complexity.
- Data type dependency: Depending on the type of data, noise accumulation varies. Text accumulates the least, whereas exhibits the fastest, followed by images. This implies that noise is affected by data structures and encoding, with audio probably requiring more intricate or numerous procedures.
- Bootstrapping threshold: The bootstrapping threshold is indicated by the red dashed line. The largest file size that can be processed without bootstrapping is indicated by the intersection of each noise curve with this line. Compared to text, audio and image data reach this limit earlier, suggesting that they require more frequent bootstrapping.

Figure 8 illustrates the impact of applying bootstrapping to mitigate noise growth in HE. It plots a smoothed representation of the noise level against file size for different data types, this time with bootstrapping enabled:

- Controlled noise: In contrast to Figure 7, the noise exhibits an increasing trend but remains under control. The "zig-

zag" pattern reflects cycles of noise increase and decrease; noise builds up until the bootstrapping threshold is reached, at which point it resets.

- Bootstrapping interval: The graph implicitly shows the effect of the bootstrapping interval. The steeper the upward slope, the faster the noise accumulates and the more frequently bootstrapping needs to be performed to maintain the noise level below the threshold.

Comparing Figures 7 and 8 highlights the effectiveness of bootstrapping. While noise grows exponentially without bootstrapping, bootstrapping can maintain it at a quasi-stable level.

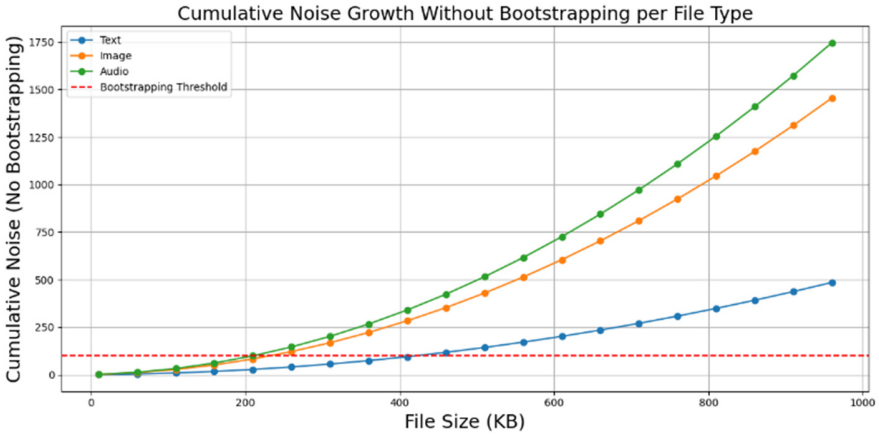

Fig. 7. Cumulative noise growth without bootstrapping per file type.

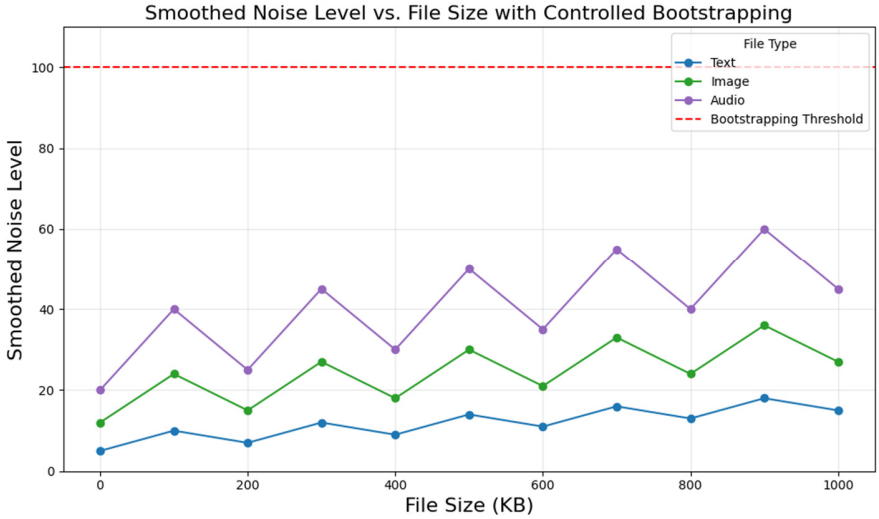

Fig. 8. Smoothed noise level vs. file size with bootstrapping enabled to control noise growth.

V. CONCLUSION AND FUTURE WORK

Adaptive User-guided Resilient Approach using Cheon-Kim-Kim-Song (AURA-CKKS) enables Homomorphic Encryption (HE) of diverse data types, including text, images, and audio, via a secure, structured pipeline. This pipeline offers user-defined configurations, vectorization, CKKS-based encoding, encryption, homomorphic processing, and decryption. The main advantage of AURA-CKKS is its adaptive noise management, which maintains result fidelity during intricate calculations by fusing bootstrapping tests with noise-reduction strategies. According to the test results, AURA-CKKS outperforms standard CKKS, achieving up to 46% increased bootstrapping efficiency, over 51% faster processing, and approximately 39% lower memory usage. These improvements demonstrate its scalability and dependability for encrypted applications that are sensitive to noise.

Future development will add Multi-Party Computation (MPC) to AURA-CKKS to enable safe, cooperative processing, which is perfect for federated learning, privacy-preserving analytics, and sensitive industries such as healthcare and finance. Additionally, the framework will be adapted for real-time and streaming data (such as time series, sensor feeds, and video) to enable low latency encrypted processing in

Internet of Things (IoT), edge computing, and multimedia systems.

Although AURA-CKKS demonstrates significant improvements in processor speed, memory utilization, and bootstrapping efficiency, still has some drawbacks. For example, the performance in high-complexity jobs may be impacted if managing low noise thresholds for large files results in an increased bootstrapping frequency. Furthermore, widespread adoption in real-world systems may be hampered by the technical complexity of user-guided controls and dynamic bootstrapping. Finally, even though text, photos, and music are included in the current experimental scope, more study is required to assess performance with additional types of data, such as video, structured databases, and real-time streams. A major focus of our future research will be addressing these limitations by investigating optimization techniques that balance performance and noise thresholds, streamlining implementation for wider adoption, and extending the framework's functionality across various application domains.

REFERENCES

[1] J. H. Cheon, A. Kim, M. Kim, and Y. 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, 2017, pp. 409–437, https://doi.org/10.1007/978-3-319-70694-8_15.

[2] J.-P. Bossuat, C. Mouchet, J. Troncoso-Pastoriza, and J.-P. Hubaux, "Efficient Bootstrapping for Approximate Homomorphic Encryption with Non-sparse Keys," in *Advances in Cryptology – EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Zagreb, Croatia, 2021, pp. 587–617, https://doi.org/10.1007/978-3-030-77870-5_21.

[3] P. N. Duong and H. Lee, "Pipelined Key Switching Accelerator Architecture for CKKS-Based Fully Homomorphic Encryption," *Sensors*, vol. 23, no. 10, May 2023, Art. no. 4594, https://doi.org/10.3390/s23104594.

[4] Y. Bae, J. H. Cheon, W. Cho, J. Kim, and T. Kim, "META-BTS: Bootstrapping Precision Beyond the Limit," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, New York, NY, USA, 2022, pp. 223–234, https://doi.org/10.1145/3548606.3560696.

[5] Y. Bae, J. H. Cheon, J. Kim, and D. Stehlé, "Bootstrapping Bits with CKKS," in *Advances in Cryptology – EUROCRYPT 2024: 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Part II*, Zurich, Switzerland, 2024, pp. 94–123, https://doi.org/10.1007/978-3-031-58723-8_4.

[6] J.-W. Lee, E. Lee, Y. Lee, Y.-S. Kim, and J.-S. No, "High-Precision Bootstrapping of RNS-CKKS Homomorphic Encryption Using Optimal Minimax Polynomial Approximation and Inverse Sine Function," in *Advances in Cryptology – EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Part I*, Zagreb, Croatia, 2021, pp. 618–647, https://doi.org/10.1007/978-3-030-77870-5_22.

[7] K. Han and D. Ki, "Better Bootstrapping for Approximate Homomorphic Encryption," in *Topics in Cryptology – CT-RSA 2020: The Cryptographers' Track at the RSA Conference 2020*, San Francisco, CA, USA, 2020, pp. 364–390, https://doi.org/10.1007/978-3-030-40186-3_16.

[8] A. A. Badawi and Y. Polyakov, "Demystifying Bootstrapping in Fully Homomorphic Encryption." Cryptology ePrint Archive, 2023. [Online]. Available: https://eprint.iacr.org/2023/149.

[9] C. Gentry, "A fully homomorphic encryption scheme," Ph.D. dissertation, Stanford University, Stanford, CA, USA, 2009.

[10] G. K. Mahato and S. K. Chakraborty, "A Comparative Review on Homomorphic Encryption for Cloud Security," *IETE Journal of Research*, vol. 69, no. 8, pp. 5124–5133, Sep. 2023, https://doi.org/10.1080/03772063.2021.1965918.

[11] V. Kadykov, A. Levina, and A. Voznesensky, "Homomorphic Encryption within Lattice-Based Encryption System," *Procedia Computer Science*, vol. 186, pp. 309–315, Jan. 2021, https://doi.org/10.1016/j.procs.2021.04.149.

[12] D. B. Salvakkam and R. Pamula, "Design of fully homomorphic multikey encryption scheme for secured cloud access and storage environment," *Journal of Intelligent Information Systems*, vol. 62, no. 3, pp. 641–663, Jun. 2024, https://doi.org/10.1007/s10844-022-00715-7.

[13] H. Patel, "Fully Homomorphic Encryption: Revolutionizing Payment Security," *International Journal of Scientific Research in Computer Science, Engineering and Information Technology*, vol. 11, no. 2, pp. 2379–2396, Mar. 2025, https://doi.org/10.32628/CSEIT25112706.

[14] H. Huang and H. Zong, "Secure matrix multiplication based on fully homomorphic encryption," *The Journal of Supercomputing*, vol. 79, no. 5, pp. 5064–5085, Mar. 2023, https://doi.org/10.1007/s11227-022-04850-4.

[15] L. Sadeghikhorami and A. A. Safavi, "Secure distributed Kalman filter using partially homomorphic encryption," *Journal of the Franklin Institute*, vol. 358, no. 5, pp. 2801–2825, Mar. 2021, https://doi.org/10.1016/j.jfranklin.2020.08.048.

[16] S. Medileh *et al.*, "A Multi-Key with Partially Homomorphic Encryption Scheme for Low-End Devices Ensuring Data Integrity," *Information*, vol. 14, no. 5, May 2023, Art. no. 263, https://doi.org/10.3390/info14050263.

[17] Y. Fang *et al.*, "Enhancing paillier to fully homomorphic encryption with semi-honest TEE," *Peer-to-Peer Networking and Applications*, vol. 17, no. 5, pp. 3476–3488, Sep. 2024, https://doi.org/10.1007/s12083-024-01752-5.

[18] K. Rajeshkumar, C. Ananth, and N. Mohananthini, "Blockchain-Assisted Homomorphic Encryption Approach for Skin Lesion Diagnosis using Optimal Deep Learning Model," *Engineering, Technology & Applied Science Research*, vol. 13, no. 3, pp. 10978–10983, Jun. 2023, https://doi.org/10.48084/etasr.5594.

[19] M. Wu, X. Zhao, and W. Song, "Bootstrapping Optimization Techniques for the FINAL Fully Homomorphic Encryption Scheme," *Information*, vol. 16, no. 3, Mar. 2025, Art. no. 200, https://doi.org/10.3390/info16030200.

[20] A. El-Yahyaoui and M. D. Ech-Cherif El Kettani, "A Verifiable Fully Homomorphic Encryption Scheme for Cloud Computing Security," *Technologies*, vol. 7, no. 1, Mar. 2019, Art. no. 21, https://doi.org/10.3390/technologies7010021.

[21] "Project Gutenberg: Free eBooks." Project Gutenberg. https://www.gutenberg.org/.

[22] J. Fan and F. Vercauteren, "Somewhat Practical Fully Homomorphic Encryption." 2012. [Online]. Available: https://eprint.iacr.org/2012/144.

[23] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(Leveled) fully homomorphic encryption without bootstrapping," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, New York, NY, USA, 2012, pp. 309–325, https://doi.org/10.1145/2090236.2090262.