PixCrypt: Fast Fine-Grained FHE with Range-Aware Caching

Chao Wang[1*], Shubing Yang[2*], Xiaoyan Sun[1], Yan Bai[2], Jun Dai[1✉], and Dongfang Zhao[2✉]

[1] Worcester Polytechnic Institute, Worcester, MA 01609, USA
{cwang17,xsun7,jdai}@wpi.edu
[2] University of Washington, Seattle, WA 98195, USA
{sueyoung,yanb}@uw.edu, dzhao@cs.washington.edu

Abstract. Many analytics tasks require secure computation over encrypted data. In particular, fine-grained data such as pixel-level images require higher precision, as every pixel can directly affect outcomes in tasks like tumor segmentation and anomaly detection. While Multi-Party Computation (MPC) is interactive, Differential Privacy (DP) protects only aggregate values, and Partially Homomorphic Encryption (PHE) lacks multiplicative support, none of them can efficiently handle fine-grained data analytics. Fully Homomorphic Encryption (FHE) uniquely enables arbitrary operations on encrypted pixels but remains computationally expensive, posing significant challenges for both software and hardware accelerators. We present PixCrypt, a caching-based acceleration mechanism for fine-grained fully homomorphic encryption. PixCrypt replaces expensive fresh ciphertext generation with cache retrieval and coefficient-level operations across CKKS, BFV, and BGV, while randomized reconstruction ensures that ciphertexts do not repeat. Its linear noise growth reduces the need for bootstrapping and lowers NTT load, improving hardware accelerator efficiency. This design yields up to 35× faster fine-grained encryption and maintains IND-CPA (Indistinguishability under Chosen Plaintext Attack) security. Experiments on five real-world pixel-level image processing tasks show that PixCrypt significantly improves the practicality of FHE for privacy-preserving analytics.

Keywords: Fully Homomorphic Encryption · Privacy-Preserving Computation · Secure Image Processing.

1 Introduction

The confidentiality of medical data, particularly diagnostic images stored in healthcare databases, faces severe risks under escalating cyberattacks. High-profile incidents such as the 2024 data leak and the 2023 ransomware attack on Lehigh Valley Health Network expose critical vulnerabilities in existing database and

* Chao Wang and Shubing Yang contributed equally to this work.
✉ Jun Dai and Dongfang Zhao are the corresponding authors.

imaging management systems. Many encryption methods reduce computation by using coarse processing, but this sacrifices pixel-level access. In medical imaging, **fine-grained** operations are essential for tasks like noise reduction, anomaly detection, and spatial analysis. Without this granularity, system effectiveness and clinical value are both compromised.

Existing secure image analysis techniques have key limitations: MPC requires distributed coordination [35], DP adds nondeterministic noise [13], and PHE lacks full computational expressiveness [3]. Fully Homomorphic Encryption (FHE), by contrast, offers a non-interactive design, supports diverse computation types, and provides strong semantic security, making it well suited for encrypted fine-grained data analysis.

Despite its expressiveness, FHE remains difficult to deploy in practice due to high computational cost. Naive pixel-level encryption and evaluation introduce prohibitive latency and memory usage, especially for high-resolution images or iterative operations. These constraints make direct FHE solutions impractical for real-time or resource-limited medical imaging, motivating the need for more efficient and scalable methods. We focus on CKKS, BFV, and BGV due to their suitability for fine-grained arithmetic, while excluding TFHE, which is optimized for Boolean circuits and does not natively support floating-point arithmetic required for common image processing operations [12].

To address the substantial performance bottleneck of fine-grained FHE in encrypted image processing, we propose PixCrypt, an optimization protocol that leverages caching mechanisms to significantly accelerate encrypted computations while maintaining fine-grained operational granularity. PixCrypt adopts two specialized caching strategies: square-root-based using a hybrid decomposition (multiplication and addition) based on square roots and parity-based using an additive decomposition into even components plus a parity bit. By reusing these pre-encrypted values during pixel-wise computation, PixCrypt significantly reduces redundant encryption operations and improves overall processing efficiency.

We implement PixCrypt on CKKS, BFV, and BGV, and evaluate it on the USC-SIPI [1] dataset across five pixel-level tasks: mean filtering, image matching, ciphertext watermarking, segmentation, and binary classification. PixCrypt delivers up to 35× faster per image encryption and preserves IND-CPA security. The key insights and contributions are summarized as follows:

- We propose a caching-based protocol that accelerates homomorphic encryption for fine-grained data through square-root-based and parity-based caching, which offer different trade-offs between offline precomputation and online encryption, while our controlled-noise design further reduces the reliance on costly bootstrapping. (Section 3.1 and 3.3)
- We design a randomization layer that adds noise during ciphertext reconstruction to ensure IND-CPA security while enabling multiple computations on the same encrypted data (Section 3.2). We formally prove that PixCrypt's IND-CPA security is guaranteed by the security of the underlying homomorphic encryption schemes. (Section 3.5)

– We conduct extensive evaluations on four distinct types of image data and five real-world pixel-level tasks and demonstrate substantial improvements in encryption efficiency across diverse scenarios, such as filtering, watermarking, segmentation and classification. (Section 4.4)

2 Related Work and Preliminaries

2.1 FHE and Acceleration

Fully Homomorphic Encryption (FHE) enables computation on encrypted data without requiring decryption, a concept first proposed by Rivest [29] and practically realized by Gentry [19]. Subsequent research has focused on enhancing its efficiency and usability [7], leading to several widely used schemes including BGV [10], BFV [15,9], CKKS [11], and TFHE [12]. These schemes differ in their arithmetic support and application contexts: TFHE excels in boolean circuits with fast bootstrapping [32], CKKS supports approximate real/complex arithmetic for ML tasks, and BFV/BGV focus on exact integer operations. While they share similar parameter foundations [6], each offers trade-offs in circuit depth, data types, and performance. Notably, BGV handles deeper circuits without bootstrapping, and BFV achieves high efficiency in certain tasks, further boosted by optimizations such as RNS techniques [21] and the BASALISC accelerator [17].

CKKS has particular traction in privacy-preserving machine learning [8] for its support of approximate arithmetic over $\mathbb{C}^{N/2}$ via batching in $\mathbb{Z}_Q[X]/(X^N+1)$. This scheme balances precision and performance, with parameters like ring dimension N and modulus Q (Table 1) critically affecting efficiency and security [24].

Table 1. Parameters and their definitions in the CKKS scheme

Symbol	Definition
N	The ring dimension
Q	The ciphertext modulus
P	The auxiliary modulus (for relinearization)
L	Maximum level (multiplicative)
ℓ	Current (multiplicative) level of a ciphertext
evk_{mul}	Evaluation key (evk) for Multiplications
Δ	Scaling factor of a CKKS plaintext
λ	Security parameter of a given CKKS instance

To improve the performance of homomorphic encryption, various acceleration techniques have been proposed. These include memory-aware bootstrapping for CKKS [5,4], gadget decomposition for optimizing bottleneck operations [18], and hardware-level enhancements such as the reconfigurable FFT/NTT hardware [34], FHE for CGRA [23], Meta-OP unified operator [26], accelerator for fast matrix-vector product [28], accelerator for TFHE [22], accelerating FHE with processing in-memory [20] and CUDA-optimized libraries [33]. Beyond FHE, efforts to speed up Partially Homomorphic Encryption (PHE) have also progressed. For instance, Rache [31] uses parallel caching to accelerate PHE for outsourced cloud databases. However, its scope is limited to PHE schemes like Paillier [27] and Symmetria [30].

2.2 Provable Security

To reason about the security of a cryptographic scheme, we specify the *security goal*, *threat model*, and *assumptions*. The security goal states what the scheme must protect; the threat model defines the adversary's capabilities; and assumptions describe the computational limits and primitives underlying the scheme.

We adopt *indistinguishability under chosen-plaintext attack* (IND-CPA). The adversary may request polynomially many encryptions of chosen plaintexts and then submit two equal-length challenge messages m_0 and m_1. The challenger samples a uniformly random bit $b \leftarrow \{0,1\}$ and fresh encryption randomness ρ, and returns the challenge ciphertext $c^* \leftarrow \mathsf{Enc}_{pk}(m_b;\rho)$. After receiving c^*, the adversary outputs a bit b'. The adversary's IND-CPA advantage is formally defined as

$$\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\Pi,\mathcal{A}}(\lambda) = \left|\Pr[b' = b] - \frac{1}{2}\right|.$$

A scheme Π is IND-CPA secure if $\mathsf{Adv}^{\mathsf{IND\text{-}CPA}}_{\Pi,\mathcal{A}}(\lambda)$ is negligible in the security parameter λ for every probabilistic polynomial-time adversary $\mathcal{A}$. A function $\mu(\lambda)$ is negligible if, for every positive polynomial $p(\lambda)$, there exists λ_0 such that $\mu(\lambda) < 1/p(\lambda)$ for all $\lambda \geq \lambda_0$. We use the standard fact that a polynomially bounded sum of negligible functions remains negligible. This fact forms the basis for proving that our construction satisfies IND-CPA security in Section 3.5.

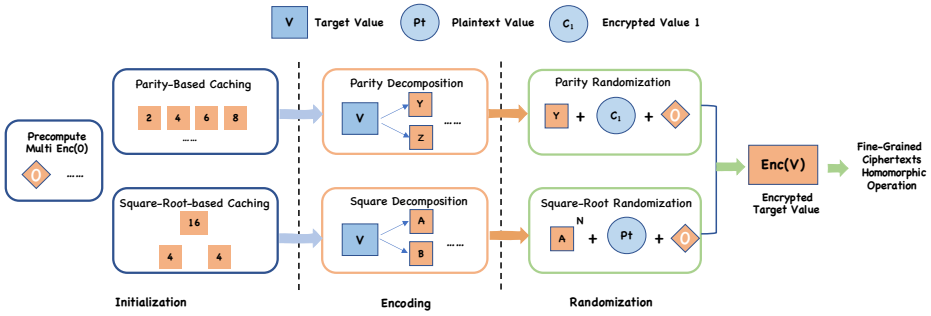

Fig. 1. Overview of PixCrypt. Parity-based and square-root-based caching accelerate scalar homomorphic encryption by combining plaintext with cached ciphertexts and injecting randomized noise to construct new ciphertexts, while preserving IND-CPA security. Blue denotes plaintext; orange denotes ciphertext.

3 Proposed Method

Our approach accelerates fine-grained data encryption under fully homomorphic encryption (FHE). While schemes such as CKKS, BFV, and BGV enable encrypted-domain computation, they incur substantial cost when encrypting individual data elements independently. To mitigate this bottleneck, we introduce two caching strategies that reduce each element encryption cost (Figure 1).

The design of caching layer is presented in Section 3.1. The randomization layer is described in Section 3.2, and the compliance of our approach with IND-CPA security is discussed in Section 3.5.

3.1 Caching Layer

Many real-world applications require fine-grained FHE operations on data with bounded integer values, such as sensor readings, rating scores, categorical variables, and digital images. A key observation is that these values are drawn from a small, finite domain (e.g., $[0, 255]$ for 8-bit data), and encryption is repeatedly applied to individual elements from this limited set. We leverage this bounded and discrete structure to accelerate encryption by precomputing and caching a compact set of "basis" ciphertexts offline, then synthesizing encryptions of arbitrary values through lightweight homomorphic operations on these cached elements. This approach trades off between cache size and online computation cost: larger caches allow purely additive synthesis with minimal computation, whereas smaller caches require extra multiplicative steps but significantly reduce storage overhead. We study two instantiations of this idea: one using an additive decomposition into even components plus a parity bit, and another using a multiplicative–additive decomposition based on square roots.

Parity-based Caching This method adopts an *additive decomposition* viewpoint. Let $\mathcal{D} = \{0, 1, \ldots, M\}$ be a bounded integer domain (e.g., $M = 255$ for 8-bit pixels). Fix a public-key FHE scheme $\Pi = (\mathrm{Setup}, \mathrm{KeyGen}, \mathrm{Enc}, \mathrm{Dec}, \mathrm{Eval})$ that supports ciphertext–ciphertext addition and either ciphertext–plaintext addition or, alternatively, ciphertext–ciphertext addition with cached small constants. We write $\mathrm{pt}(\cdot)$ for the scheme's plaintext encoding (with a fixed slot layout and, for CKKS, a fixed scale), and use $\oplus, \ominus$ for homomorphic addition and subtraction. We precompute only *even* encryptions together with an encryption of 1 and a small pool of fresh zeros: $\mathcal{C}_{\mathrm{even}} = \{\mathrm{Enc}(0), \mathrm{Enc}(2), \ldots, \mathrm{Enc}(2\lfloor M/2 \rfloor)\}$, $\mathcal{C}_1 = \mathrm{Enc}(1)$, and $\mathcal{Z} = \{\mathrm{Enc}_{\mathrm{rand}}(0)_1, \ldots, \mathrm{Enc}_{\mathrm{rand}}(0)_{N_z}\}$, where $\mathcal{Z}$ denotes a shared pool of N_z fresh, independent encryptions of zero used for re-randomization in both caching strategies, and N_z is a system parameter controlling the pool size. For any plaintext $t \in \mathcal{D}$, we decompose it using parity as $t = 2v + \delta$ where $v = \lfloor t/2 \rfloor$ and $\delta = t - 2v \in \{0, 1\}$. We then construct the encryption by retrieving $\mathrm{Enc}(2v)$ from $\mathcal{C}_{\mathrm{even}}$ and sampling a fresh $\mathrm{Enc}(0)$ from $\mathcal{Z}$ for randomization.

The output of Algorithm 1 serves as the input to Algorithm 3.

Square-Root-based Caching In contrast, this method uses a *multiplicative–additive hybrid* decomposition. Let $\mathcal{D} = \{0, 1, \ldots, M\}$ be a bounded integer domain (e.g., $M = 255$ for 8-bit pixels). Fix a public-key FHE scheme $\Pi = (\mathrm{Setup}, \mathrm{KeyGen}, \mathrm{Enc}, \mathrm{Dec}, \mathrm{Eval})$ that supports ciphertext–ciphertext multiplication, ciphertext–plaintext multiplication, and ciphertext–plaintext addition. We write $\mathrm{pt}(\cdot)$ for the scheme's plaintext encoding (with a fixed slot layout and, for CKKS, a fixed scale), and use $\oplus, \ominus, \odot$ for homomorphic addition, subtraction and multiplication. We precompute only *square root* encryptions together with a

Algorithm 1: Parity-based Caching

Input: Value Collection $\mathcal{D} = \{0, 1, \ldots, M\}$; FHE scheme $\Pi = (\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Dec}, \text{Eval})$; Plaintext $t \in \mathcal{D}$

Output: *CachedList*

```
// Precomputation phase (done once)
```

if *cache not initialized* **then**

- $\mathcal{C}_{\text{even}} \leftarrow \{\text{Enc}(0), \text{Enc}(2), \text{Enc}(4), \ldots, \text{Enc}(2\lfloor M/2 \rfloor)\}$;
- $\mathcal{C}_1 \leftarrow \text{Enc}(1)$ `// reusable encryption of 1`
- $\mathcal{Z} \leftarrow \{\text{Enc}_{\text{rand}}(0)_1, \text{Enc}_{\text{rand}}(0)_2, \ldots, \text{Enc}_{\text{rand}}(0)_{N_z}\}$ `// shared zero pool for re-randomization`

```
// Parity decomposition
```

$v \leftarrow \lfloor t/2 \rfloor$;

$\delta \leftarrow t - 2v$ `//` $\delta \in \{0, 1\}$

```
// Retrieve base even encryption
```

$C_{2v} \leftarrow$ lookup $\text{Enc}(2v)$ from $\mathcal{C}_{\text{even}}$;

return $\{\mathcal{C}_{\text{even}}, \mathcal{C}_1, \mathcal{Z}\}$;

small pool of N_z fresh zeros: $\mathcal{C}_{\text{sqrt}} = \{\text{Enc}(0), \text{Enc}(1), \ldots, \text{Enc}(\lfloor \sqrt{M} \rfloor)\}$ and $\mathcal{Z} = \{\text{Enc}_{\text{rand}}(0)_1, \ldots, \text{Enc}_{\text{rand}}(0)_{N_z}\}$. Additionally, we precompute plaintext encodings for all square root values and remainders: $\mathcal{P}_{\text{sqrt}} = \{\text{pt}(0), \text{pt}(1), \ldots, \text{pt}(\lfloor \sqrt{M} \rfloor)\}$ and $\mathcal{P}_{\text{rem}} = \{\text{pt}(0), \text{pt}(1), \ldots, \text{pt}(2\lfloor \sqrt{M} \rfloor)\}$. For any $v \in \mathcal{D}$, express the square decomposition $v = k^2 + r$, where $k = \lfloor \sqrt{v} \rfloor$ and $r = v - k^2 \in \{0, 1, \ldots, 2\lfloor \sqrt{M} \rfloor\}$. To synthesize an encryption of v, retrieve $\text{Enc}(k) \in \mathcal{C}_{\text{sqrt}}$ and set

$$C_v = \big(\text{Enc}(k) \odot \text{pt}(k)\big) \oplus \text{pt}(r) \oplus Z, \qquad Z \leftarrow \mathcal{Z}.\text{pop}(), \tag{1}$$

where the multiplication $\text{Enc}(k) \odot \text{pt}(k)$ computes k^2 homomorphically, followed by addition of the remainder r and fresh zero re-randomization.

The output of Algorithm 2 serves as the input to Algorithm 4.

3.2 Randomization Layer

If ciphertexts are exposed and attackers have prior knowledge about plaintext distributions or access to an encryption oracle, encryption schemes become susceptible to chosen-plaintext attacks (CPA). Thus, it is essential to introduce randomness each time a cached ciphertext is used.

Parity-based Randomization Given a plaintext value t, we decompose it based on its parity as $t = 2v + \delta$ where $v = \lfloor t/2 \rfloor$ and $\delta \in \{0, 1\}$ represents the remainder (i.e., whether t is even or odd).

To generate a fresh ciphertext encrypting the same value t, we reconstruct it from cached components: precomputed even ciphertexts from cache $\mathcal{C}_{\text{even}}$, a unit ciphertext $\mathcal{C}_1 = \text{Enc}(1)$, and fresh zero ciphertexts from cache $\mathcal{Z}$. The

Algorithm 2: Square-Root-based Caching

Input: Domain $\mathcal{D} = \{0, 1, \ldots, M\}$; FHE scheme $\Pi = (\text{Setup}, \text{KeyGen}, \text{Enc}, \text{Dec}, \text{Eval})$; Plaintext $t \in \mathcal{D}$
Output: *CachedList*
Initialize HE context for Π (parameters, keys, encoder/decoder, evaluator);
Build caches;
for $k = 0$ **to** $\lfloor\sqrt{M}\rfloor$ **do**
  $\mathcal{C}_{\text{sqrt}}[k] \leftarrow \text{Enc}(k)$; `// ciphertexts of` $0, \ldots, \lfloor\sqrt{M}\rfloor$
for $k = 0$ **to** $\lfloor\sqrt{M}\rfloor$ **do**
  $\mathcal{P}_{\text{sqrt}}[k] \leftarrow \text{pt}(k)$; `// plaintext encodings`
for $r = 0$ **to** $2\lfloor\sqrt{M}\rfloor$ **do**
  $\mathcal{P}_{\text{rem}}[r] \leftarrow \text{pt}(r)$; `// plaintext remainders`
for $i = 1$ **to** N_z **do**
  $\mathcal{Z}[i] \leftarrow \text{Enc}_{\text{rand}}(0)$; `// independent encryptions of` 0
return $\mathit{CachedList} = \{\mathcal{C}_{\text{sqrt}}, \mathcal{P}_{\text{sqrt}}, \mathcal{P}_{\text{rem}}, \mathcal{Z}\}$;

Algorithm 3: Parity-Based Randomization

Input: Plaintext $t \in \{0, 1, \ldots, M\}$; Cache $\mathcal{C}_{\text{even}}, \mathcal{C}_1$; Zero pool $\mathcal{Z} = \{\text{Enc}_{\text{rand}}(0)_1, \ldots, \text{Enc}_{\text{rand}}(0)_{N_z}\}$.
Output: Randomized ciphertext C' such that $\mathsf{Dec}(C') = t$.

$v \leftarrow \lfloor t/2 \rfloor$;
$\delta \leftarrow t - 2v$;
$C_{\text{base}} \leftarrow \mathcal{C}_{\text{even}}[v]$;
$Z \leftarrow \mathcal{Z}.pop()$;
if $\delta = 0$ **then**
  $C' \leftarrow C_{\text{base}} \oplus Z$;
else
  $C' \leftarrow C_{\text{base}} \oplus \mathcal{C}_1 \oplus Z$;
return C';

reconstruction process depends on the parity bit δ:

$$C' \leftarrow \begin{cases} \text{Enc}(2v) \oplus Z, & \text{if } \delta = 0 \text{ (even)}, \\ \text{Enc}(2v) \oplus \mathcal{C}_1 \oplus Z, & \text{if } \delta = 1 \text{ (odd)}, \end{cases}$$

where Z is a freshly sampled encryption of zero from $\mathcal{Z}$. A fresh zero is popped from the precomputed pool for each reconstruction. This approach preserves the plaintext value while introducing fresh encryption randomness through the zero ciphertext, as shown in Algorithm 3. The method leverages the additive homomorphic property of the encryption scheme to combine cached ciphertexts with fresh randomness efficiently.

Algorithm 4: Square-Root-Based Randomization

Input: Domain $\mathcal{D} = \{0, 1, \ldots, M\}$;
Target plaintext $t \in \mathcal{D}$;
Ciphertext cache $\mathcal{C}_{\text{sqrt}} = \{\mathsf{Enc}(k)\}_{k=0}^{\lfloor\sqrt{M}\rfloor}$;
Plaintext tables:
$\mathcal{P}_{\text{sqrt}} = \{\mathsf{pt}(k)\}_{k=0}^{\lfloor\sqrt{M}\rfloor}$,
$\mathcal{P}_{\text{rem}} = \{\mathsf{pt}(r)\}_{r=0}^{2\lfloor\sqrt{M}\rfloor}$;
Zero pool $\mathcal{Z} = \{\text{Enc}_{\text{rand}}(0)_1, \ldots, \text{Enc}_{\text{rand}}(0)_{N_z}\}$.
Output: Ciphertext C_t such that $\mathsf{Dec}(C_t) = t$.

$k \leftarrow \lfloor\sqrt{t}\rfloor$;
$r \leftarrow t - k^2$

$C_k \leftarrow \mathcal{C}_{\text{sqrt}}[k]$ $P_k \leftarrow \mathcal{P}_{\text{sqrt}}[k]$ $B_{\text{base}} \leftarrow C_k \odot P_k$

if scheme is CKKS **then**
  $\mathsf{RescaleTo}(B_{\text{base}}, \texttt{target\_scale})$;

$P_r \leftarrow \mathcal{P}_{\text{rem}}[r]$ $B_{\text{sum}} \leftarrow B_{\text{base}} \oplus P_r$;

$Z \leftarrow \mathcal{Z}.pop()$;
$C_t \leftarrow B_{\text{sum}} \oplus Z$;

return C_t;

Square-Root-based Randomization. Given a target value $t \in \mathcal{D} = \{0, \ldots, M\}$ and precomputed resources including a square root cache $\mathcal{C}_{\text{sqrt}} = \{\text{Enc}(0), \text{Enc}(1), \ldots, \text{Enc}(\lfloor\sqrt{M}\rfloor)\}$, plaintext tables $\mathcal{P}_{\text{sqrt}} = \{\text{pt}(0), \ldots, \text{pt}(\lfloor\sqrt{M}\rfloor)\}$ and $\mathcal{P}_{\text{rem}} = \{\text{pt}(0), \ldots, \text{pt}(2\lfloor\sqrt{M}\rfloor)\}$, and a zero-pool $\mathcal{Z} = \{\text{Enc}_{\text{rand}}(0)_1, \ldots, \text{Enc}_{\text{rand}}(0)_{N_z}\}$, we construct a fresh ciphertext encrypting t through square-root decomposition. As shown in Algorithm 4, the construction begins by decomposing the target as $t = k^2 + r$ where $k = \lfloor\sqrt{t}\rfloor$ and $r = t - k^2$. We then compute k^2 homomorphically using the precomputed encryption and plaintext values as $B \leftarrow \mathcal{C}_{\text{sqrt}}[k] \odot \mathcal{P}_{\text{sqrt}}[k]$. For CKKS schemes, we perform scale management by rescaling B to the target scale before proceeding. Next, we incorporate the remainder through plaintext addition, $C_t \leftarrow B \oplus \mathcal{P}_{\text{rem}}[r]$. Finally, we achieve re-randomization by sampling a fresh zero encryption Z from the zero-pool $\mathcal{Z}$ and adding it to the result, $C_t \leftarrow C_t \oplus Z$. This approach efficiently constructs target ciphertexts by leveraging precomputed square values up to $\sqrt{M}$, while ensuring cryptographic freshness through the zero-pool mechanism. The square-root decomposition reduces the computational overhead compared to direct homomorphic arithmetic while maintaining the desired plaintext value.

Example. To encrypt $t = 15$ where $\mathcal{D} = \{0, \ldots, 16\}$: **Parity-based**: Decompose $15 = 2 \cdot 7 + 1$, then compute $\text{Enc}(15) = \text{Enc}(14) \oplus \text{Enc}(1) \oplus Z$ using only additions. **Square-root**: Decompose $15 = 3^2 + 6$, then compute $\text{Enc}(15) = (\text{Enc}(3) \odot \text{pt}(3)) \oplus \text{pt}(6) \oplus Z$ using multiplication and addition. Here Z is a fresh $\text{Enc}(0)$ for randomization and $\text{pt}(\cdot)$ denotes plaintext encoding.

3.3 Bootstrapping Reduction via Plaintext-Oriented Encoding

Bootstrapping is essential but costly in FHE, triggered when ciphertext noise exceeds a threshold. We propose a strategy that fundamentally reduces noise growth by shifting arithmetic computation to the plaintext domain. Instead of using ciphertext-ciphertext multiplications, our method combines cached ciphertexts with low-noise plaintext operations, keeping noise growth strictly additive. The key advantage is that while our approach introduces additional noise, this growth remains predictable and tightly bounded—linear (pt–ct) rather than multiplicative (ct–ct) noise accumulation in traditional methods.

Let d denote the operational depth, i.e., the maximum number of sequential homomorphic operations supported before bootstrapping is required, and $O(\cdot)$ denote standard asymptotic (big-O) notation. Let $\mathsf{Enc}_{pk}(m)$ denote the encryption with initial noise level σ_0, where t denotes the number of sequential homomorphic operations. σ_0 is the initial noise level of a fresh ciphertext, Δ_{add} and Δ_{mult} denote the per-operation noise increments for plaintext-ciphertext addition and ciphertext-ciphertext multiplication respectively, ϵ_j is the noise perturbation at step j, and $\mathsf{threshold}$ denotes the maximum noise level before bootstrapping is triggered. The noise evolution follows:

$$\begin{aligned}\mathsf{noise}_{\text{ours}}(t) &= \sigma_0 + \sum_{j=1}^{t} \Delta_{\text{add},j} + \underbrace{O(\log M)}_{\text{controlled overhead}} \\ &= O(\sigma_0 + t \cdot \Delta_{\text{add}} + \log M),\end{aligned}$$

$$\mathsf{noise}_{\text{baseline}}(t) = \sigma_0 \cdot \prod_{j=1}^{t}(1+\epsilon_j) \;\le\; O(\sigma_0 \cdot \Delta_{\text{mult}}^{t}).$$

Critically, our noise growth is linear in t and bounded by $O(t \cdot \Delta_{\text{add}} + \log M)$. Our method's additive noise accumulation, deterministically bounded by the worst-case pt–ct operation noise Δ_{add}, permits precise noise-budget management, unlike the multiplicative growth in baseline approaches. This controlled noise extends operational depth before bootstrapping, i.e., $d_{\text{ours}} \gg d_{\text{baseline}}$, specifically as follows:

$$d_{\text{ours}} = O\left(\frac{\mathsf{threshold}}{\Delta_{\text{add}}}\right) \gg d_{\text{baseline}} = O\left(\log_{\Delta_{\text{mult}}} \mathsf{threshold}\right).$$

3.4 Complexity Analysis

We compare the computational and space complexity of the proposed caching strategies with the baselines. Let n denote the number of pixels, M the domain size (typically 255 for 8-bit data), and t_e the cost of one encryption, typically $O(N \log N)$ for polynomial modulus degree N. Without caching, encryption costs $O(n \cdot N \log N)$, whereas Rache uses $O(\lceil \log_r M \rceil)$ space and $O(n \cdot \lceil \log_r M \rceil)$

Table 2. Comparison of complexity in terms of precomputation time, encryption time, and space.

Method	Scheme	Precomputation Time	Encryption Time	Space Complexity
Without Caching (Baseline)	CKKS / BFV / BGV	None	$O(n \cdot N \log N)$	None
Rache [31] (Baseline)	CKKS / BFV / BGV	$O(\lceil \log_r M \rceil \cdot t_e)$	$O(n \cdot \lceil \log_r M \rceil)$	$O(\lceil \log_r M \rceil)$
Parity-based (**Ours**)	CKKS / BFV / BGV	$O(M/2 \cdot t_e)$	$O(n)$	$O(M/2)$
Square-root-based (**Ours**)	CKKS / BFV / BGV	$O(\sqrt{M} \cdot t_e)$	$O(n \cdot \log \sqrt{M})$	$O(\sqrt{M})$

Note: n = number of pixels, M = domain size (e.g., 255 for 8-bit), N = polynomial modulus degree, t_e = time complexity of single encryption ($O(N \log N)$), r = radix base.

online time. Our parity-based method uses $O(M/2)$ space and $O(n)$ online time, while the square-root-based method uses $O(\sqrt{M})$ space with homomorphic multiplication during online synthesis. Table 2 summarizes these trade-offs. We instantiate these space bounds for 8, 12, and 16-bit domains in Section 4.6.

3.5 Security Analysis and Proof

In this section, we demonstrate that the modified FHE schemes, incorporating parity-based and square-root-based caching strategies, maintain IND-CPA security. Our caching mechanisms are designed to be scheme-agnostic and can be applied to various FHE schemes including BFV, BGV, and CKKS, all of which base their security on the Ring Learning with Errors (RLWE) problem. We rely on additive re-randomization: if c encrypts m, then $c \oplus \mathsf{Enc}_{pk}(0;\rho)$ is indistinguishable from a fresh encryption of m for independently sampled ρ.

Threat Model. We consider secure computation over encrypted bounded scalar values, including structured data (integers, categorical labels, scores) and quantized image data (8-bit intensities). The scenario involves outsourcing encrypted data to an untrusted server that performs computations without decryption keys. The adversary—either external or insider—has full access to ciphertexts and computational transcripts but no secret keys. While FHE enables computation on ciphertexts without exposing plaintext, it incurs significant overhead, particularly from bootstrapping. Our work optimizes encryption strategies for bounded scalar domains to minimize noise growth and reduce bootstrapping frequency, making FHE more practical while preserving security guarantees. Each zero ciphertext in $\mathcal{Z}$ is independently generated, kept private by the encrypting client, used only once, and refreshed before pool exhaustion; thus, every reconstruction incorporates fresh randomness even when Eval is deterministic.

Security of Parity-based Caching. Note that this construction is compatible with BFV, BGV, and CKKS schemes as they all operate over similar ring structures and rely on RLWE for security. Let $\Pi(m)$ denote the encryption function of a plaintext m under the base FHE scheme. For a plaintext $m = 2v + \delta$ with $v \in \{0, \ldots, \lfloor M/2 \rfloor\}$ and $\delta \in \{0, 1\}$, the cached encryption takes the form $\widetilde{\Pi}(m) = \Pi(2v) \oplus \delta \cdot \Pi(1) \oplus Z$, where $\Pi(2v)$ is from the precomputed even-cache, $\Pi(1)$ is cached, and Z is a fresh encryption of 0 sampled from the zero-pool $\mathcal{Z}$.

Assume for the sake of contradiction that the modified scheme $\widetilde{\Pi}$ with parity-based caching is not IND-CPA secure. Let $\mathrm{CPA}_X^{\mathcal{A}}$ denote the indistinguishability

experiment with scheme X. By assumption, the base scheme is IND-CPA secure with $\Pr[\mathrm{CPA}_{\Pi}^{\mathcal{A}} = 1] \leq \frac{1}{2} + \varepsilon$, where ε is negligible.

In $\widetilde{\Pi}$, each ciphertext is computed as the sum of cached $\Pi(2v)$, $\delta \cdot \Pi(1)$, and a fresh zero ciphertext Z. Each cached component $\Pi(2v)$ and $\Pi(1)$ is itself an RLWE sample indistinguishable from random, and adding an independently sampled $Z = \Pi(0; \rho)$ re-randomizes the ciphertext distribution. The determinism of Eval does not make $\widetilde{\Pi}$ deterministic, because the independently generated one-time ciphertext Z introduces fresh randomness into every output. Therefore, the challenge ciphertext distribution under $\widetilde{\Pi}$ is computationally indistinguishable from that under Π:

$$\left|\Pr[\mathrm{CPA}_{\widetilde{\Pi}}^{\mathcal{A}} = 1] - \Pr[\mathrm{CPA}_{\Pi}^{\mathcal{A}} = 1]\right| \leq \mu(\lambda). \tag{2}$$

Thus:

$$\Pr[\mathrm{CPA}_{\widetilde{\Pi}}^{\mathcal{A}} = 1] \leq \frac{1}{2} + \varepsilon + \mu(\lambda). \tag{3}$$

Since both ε and $\mu(\lambda)$ are negligible, their sum is negligible. Therefore, the probability that $\mathcal{A}$ succeeds in the $\mathrm{CPA}_{\widetilde{\Pi}}^{\mathcal{A}}$ experiment is only negligibly higher than $\frac{1}{2}$, contradicting our assumption and proving that parity-based caching is IND-CPA secure.

Security of Square-Root-based Caching. Let M be the maximum value in the bounded integer domain $\mathcal{D} = \{0, 1, \ldots, M\}$ (e.g., $M = 255$ for 8-bit pixels). The construction is compatible with BFV, BGV, and CKKS, since these schemes share similar ring structures and rely on RLWE security. Let $\Pi(m)$ denote the encryption of plaintext m under the base FHE scheme. For $v \in \mathcal{D}$, its cached encryption is

$$c = \widetilde{\Pi}(v) = \Pi(k) \odot \mathrm{pt}(k) \oplus \mathrm{pt}(r) \oplus Z, \tag{4}$$

where $v = k^2 + r$, $k = \lfloor \sqrt{v} \rfloor$, $r = v - k^2 \in \{0, 1, \ldots, 2\lfloor \sqrt{M} \rfloor\}$, and Z is a fresh zero encryption from the zero-pool $\mathcal{Z}$. Assume for contradiction that the modified scheme $\widetilde{\Pi}$ with square-root-based caching is not IND-CPA secure. Let $\mathrm{CPA}_{X}^{\mathcal{A}}$ denote the indistinguishability experiment for scheme X. The success probabilities of $\mathcal{A}$ against Π and $\widetilde{\Pi}$ are $\Pr[\mathrm{CPA}_{\Pi}^{\mathcal{A}} = 1]$ and $\Pr[\mathrm{CPA}_{\widetilde{\Pi}}^{\mathcal{A}} = 1]$, respectively. Here, λ is the security parameter of the encryption scheme, determined by the underlying RLWE dimension and unrelated to the number of image pixels. By assumption, $\Pr[\mathrm{CPA}_{\Pi}^{\mathcal{A}} = 1] \leq \frac{1}{2} + \varepsilon$, where ε is negligible.

In $\widetilde{\Pi}$, each ciphertext is formed from the cached $\Pi(k)$ by a public plaintext multiplication $\odot\, \mathrm{pt}(k)$, a plaintext addition $\oplus\, \mathrm{pt}(r)$, and a fresh zero Z. Each $\Pi(k)$ is itself an RLWE sample indistinguishable from random; the plaintext operations are public and, given k and r, message-independent; and adding an independently sampled $Z = \Pi(0; \rho)$ re-randomizes the ciphertext distribution, so repeated encryptions of the same value differ with overwhelming probability. Therefore the challenge distribution under $\widetilde{\Pi}$ is computationally indistinguishable from that under Π, and $|\Pr[\mathrm{CPA}_{\widetilde{\Pi}}^{\mathcal{A}} = 1] - \Pr[\mathrm{CPA}_{\Pi}^{\mathcal{A}} = 1]| \leq \mu(\lambda)$. Combining the above,

$$\Pr[\mathrm{CPA}_{\widetilde{\Pi}}^{\mathcal{A}} = 1] \leq \frac{1}{2} + \varepsilon + \mu(\lambda). \tag{5}$$

The last two terms are negligible: ε is negligible by the IND-CPA security of the base scheme, and $\mu(\lambda)$ is negligible by the computational indistinguishability

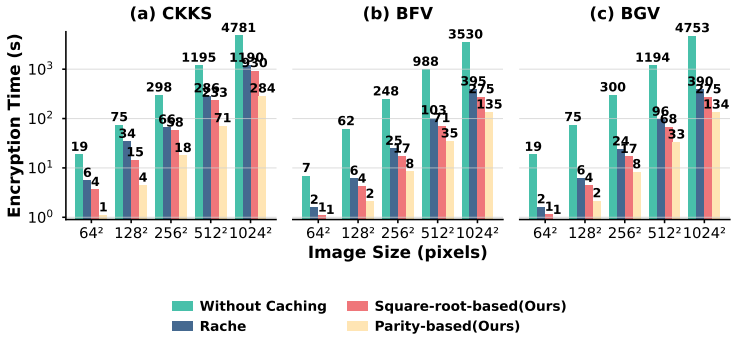

Fig. 2. Comparison of encryption time for three schemes across different image sizes.

Table 3. Comparison of encryption time(s) for three FHE schemes with varying $\log QP$.

Scheme	logN	Method	logQP 437	531	626	726	831	logN	Method	logQP 437	531	626	726	831
CKKS	12	Without Caching	18.62	28.71	38.56	48.98	60.55	14	Without Caching	79.53	125.71	172.31	220.76	275.68
		Rache	3.70	7.64	10.39	13.68	18.32		Rache	13.06	27.94	45.11	70.62	89.25
		Sqrt-based (Ours)	3.63	5.59	7.50	9.53	11.77		Sqrt-based (Ours)	15.47	24.46	33.54	42.95	53.58
		Parity-based (Ours)	**1.11**	**1.71**	**2.29**	**2.93**	**3.60**		Parity-based (Ours)	**4.73**	**7.48**	**10.25**	**13.08**	**16.40**
	13	Without Caching	38.37	59.81	81.19	103.17	127.05	15	Without Caching	260.38	346.95	369.96	492.74	622.45
		Rache	7.00	13.62	21.69	29.20	36.25		Rache	28.09	72.08	115.29	156.60	195.77
		Sqrt-based (Ours)	7.46	11.63	15.80	20.09	24.74		Sqrt-based (Ours)	50.67	67.54	71.97	95.98	121.05
		Parity-based (Ours)	**2.28**	**3.46**	**4.69**	**5.96**	**7.34**		Parity-based (Ours)	**15.48**	**20.64**	**21.99**	**29.06**	**36.98**
BFV	12	Without Caching	14.00	20.40	26.55	33.13	40.35	14	Without Caching	58.74	86.85	115.68	146.99	176.96
		Rache	1.58	2.93	4.48	5.71	7.16		Rache	5.94	12.20	18.24	26.32	33.70
		Sqrt-based (Ours)	1.10	2.04	3.12	3.97	4.99		Sqrt-based (Ours)	4.14	8.49	12.71	18.33	23.46
		Parity-based (Ours)	**0.54**	**1.00**	**1.53**	**1.94**	**2.44**		Parity-based (Ours)	**2.03**	**4.16**	**6.22**	**8.97**	**11.48**
	13	Without Caching	28.25	42.34	55.75	69.69	83.47	15	Without Caching	120.71	182.32	248.68	312.12	371.11
		Rache	2.78	5.80	8.77	11.79	15.57		Rache	11.89	26.80	41.66	54.78	67.31
		Sqrt-based (Ours)	1.94	3.99	6.11	8.20	10.85		Sqrt-based (Ours)	8.28	18.65	28.98	38.17	46.78
		Parity-based (Ours)	**0.95**	**1.98**	**3.00**	**4.01**	**5.30**		Parity-based (Ours)	**4.05**	**9.12**	**14.18**	**18.63**	**22.93**
BGV	12	Without Caching	17.01	26.86	36.77	46.85	57.49	14	Without Caching	73.34	116.55	160.50	205.56	256.51
		Rache	1.54	2.92	4.51	5.86	7.15		Rache	5.85	11.92	18.60	25.88	33.63
		Sqrt-based (Ours)	1.08	2.05	3.17	4.12	5.02		Sqrt-based (Ours)	4.11	8.38	13.07	18.20	23.64
		Parity-based (Ours)	**0.53**	**1.01**	**1.55**	**2.02**	**2.46**		Parity-based (Ours)	**2.02**	**4.09**	**6.40**	**8.91**	**11.58**
	13	Without Caching	35.33	56.06	76.79	98.16	120.20	15	Without Caching	151.69	245.38	338.29	439.94	536.78
		Rache	2.96	5.77	8.62	12.25	15.04		Rache	11.46	26.02	42.11	54.66	69.26
		Sqrt-based (Ours)	2.08	4.05	6.06	8.62	10.57		Sqrt-based (Ours)	8.06	18.33	29.57	38.91	49.13
		Parity-based (Ours)	**1.02**	**1.99**	**2.97**	**4.02**	**5.17**		Parity-based (Ours)	**3.94**	**8.95**	**14.54**	**20.04**	**23.85**

argument above. Hence, $\mathcal{A}$ succeeds in $\mathrm{CPA}^{\mathcal{A}}_{\widetilde{\Pi}}$ with probability only negligibly greater than $\frac{1}{2}$, contradicting the assumption. Therefore, the scheme with square-root-based caching is IND-CPA secure.

4 Evaluation

This section presents our experimental setup (Section 4.1), performance comparisons (Section 4.2), ablation studies (Section 4.3), real-world applications (Section 4.4), bootstrapping frequency analysis (Section 4.5), and higher-bit-depth scalability (Section 4.6).

4.1 Experimental Setup

Our implementation is based on the Lattigo library [2], an open-source Go library for lattice-based cryptography, and PixCrypt contains about 10K lines of Go code. All experiments run on a CloudLab Clemson cluster with two 32-core AMD 7542 processors at 2.9 GHz, 512 GB RAM, and 2 TB SSD, using Ubuntu 20.04 LTS. We use the USC-SIPI Image Database [1] and generate image subsets at resolutions 64×64, 128×128, 256×256, 512×512, and 1024×1024. The dataset includes multiple categories such as aerials, miscellaneous objects, sequences, and textures.

4.2 Performance Comparison

Time and Scalability We evaluate encryption efficiency across image sizes and schemes (CKKS, BFV, BGV) using the same parameters ($\log N = 12$, $\log QP = 109$; see Table 1). Each test is run three times and we report the average. Figure 2 shows encryption time on a logarithmic scale. The baseline without caching increases rapidly with image size for all schemes because each ciphertext requires fresh randomness and full polynomial generation. We use a per-value baseline because per-ciphertext cost is precisely what we reduce; SIMD packing cuts the number of encryptions, not the cost of each one. The parity-based caching method provides the best performance, achieving up to 35× speedup and consistently outperforming Rache and square-root-based caching across all schemes. Its efficiency comes from reusing pre-encrypted ciphertexts and updating only polynomial coefficients through lightweight homomorphic operations, reducing encryption to fast reconstruction with $O(1)$ cache access.

Parameter Impact on Performance We evaluate the performance of our caching methods across CKKS, BFV, and BGV by varying $\log N$ and $\log QP$, comparing four approaches: without caching, Rache, square-root-based caching, and parity-based caching. As $\log N$ increases (Figure 3), encryption time for the baseline grows rapidly due to larger ciphertexts, while square-root-based caching reduces encryptions through decomposition and parity-based caching offers the best performance via $O(1)$ ciphertext lookup. At the largest $\log N$ values, parity-based caching achieves up to 19×, 18×, and 25× speedups in CKKS, BFV, and BGV, respectively.

When varying $\log QP$ (Table 3), parity-based caching remains the fastest method across all evaluated configurations, while square-root-based caching requires less cache memory. Overall, both methods consistently accelerate encryption across schemes and parameters, and because most non-modulus parameters in Lattigo only marginally affect encryption time, these results highlight the generality and robustness of our approach.

Trade-off Analysis We summarize the trade-offs among precomputation, online time, and cache memory using CKKS at $\log N = 12$ on 64×64 images (trends

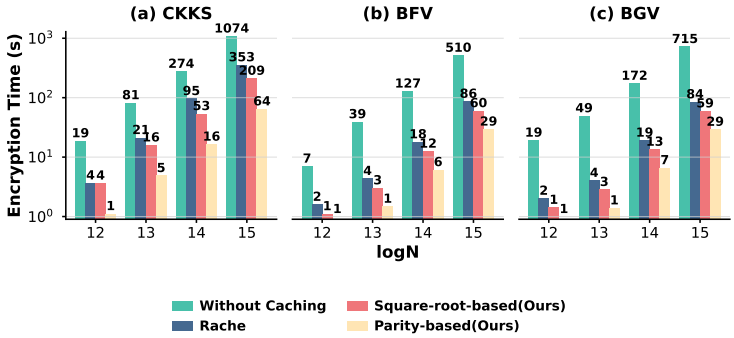

Fig. 3. Comparison of encryption time for three FHE schemes with varying LogN.

Table 4. Encryption time (s) across image types for four strategies. Parity-based achieves the lowest encryption time in all categories. The speedup is measured against the No Cache baseline.

Type	No Cache	Rache	Sqrt. (Ours)	Parity (Ours)	Speedup
Aerials	297	162	71	**19**	15.6×
Misc.	331	180	83	**25**	13.2×
Sequences	311	170	68	**18**	17.3×
Textures	281	153	68	**18**	15.6×

hold at higher resolutions). The no-caching baseline does no precomputation and is slowest online (19 s). Rache [31] adds 0.032 s precomputation and a 1 MB cache, cutting online time to 6 s. Our methods shift more work offline: parity uses 0.512 s and 16 MB to reach 1 s, while square-root uses 0.064 s and 2 MB to reach 4 s.

Impact of Image Categories To evaluate performance across different image categories, we select 200 images of size 256×256 from textures, aerial scenes, miscellaneous objects, and sequences in the USC-SIPI dataset [1], using CKKS with parity-based caching.

As shown in Table 4, PixCrypt achieves speedups ranging from 13.2× to 17.3× across all four categories, indicating stable performance across different image content.

4.3 Ablation Studies

To quantify the cost of randomization layer, we isolate ciphertext re-randomization time from the total encryption time across CKKS, BFV, and BGV under varying $\log N$, as shown in Table 5. For parity-based caching, randomization time is virtually identical to total encryption time across all schemes and $\log N$ settings (e.g., 1.01 s vs. 1.01 s for CKKS at $\log N$=12), showing that randomization accounts for most of the online reconstruction cost. For sqrt-based caching, a small gap exists (e.g., 3.74 s vs. 3.86 s for CKKS at $\log N$=12), attributable to

Table 5. Ablation study of the two layers in our method.

Scheme	Method	logN	Randomization Time (s)	Encryption Time (s)
CKKS	Sqrt-based	12	**3.74**	**3.86**
		13	22.06	22.31
		14	110.4	111.14
	Parity-based	12	**1.01**	**1.01**
		13	6.39	6.39
		14	23.33	23.33
BFV	Sqrt-based	12	**3.90**	**3.98**
		13	10.86	10.95
		14	52.49	52.62
	Parity-based	12	**1.72**	**1.72**
		13	5.03	5.03
		14	23.61	23.61
BGV	Sqrt-based	12	**3.87**	**3.95**
		13	11.69	11.85
		14	46.78	46.92
	Parity-based	12	**1.73**	**1.73**
		13	4.73	4.73
		14	7.47	7.47

the ciphertext–plaintext multiplication in the square decomposition rather than re-randomization itself.

These results show that randomization constitutes the primary encryption cost, which is necessary to ensure IND-CPA security, while other operations introduce negligible overhead.

4.4 Real-world Applications

To demonstrate that PixCrypt enables practical ciphertext-based image processing, we evaluate it on five representative tasks that require pixel-level operations, as shown in Figure 4:

Mean Filtering. We implement 3×3 mean filtering [14] on 64×64 encrypted images. As expected, the denoised output yields noticeable visual differences (MSE=218.13, PSNR=24.74). Compared to the 84.29s baseline, PixCrypt reduces encryption time to 4.65s, achieving 18× speedup, with similar gains for Gaussian filters.

Image Matching. We compute encrypted L1 distances between 128×128 images for similarity evaluation, enabling private image identification in contexts such as face recognition [25]. PixCrypt reduces encryption time by 12×, from 826.19s to 69.99s.

Ciphertext-based Watermarking. To assert ownership over encrypted media [16], we embed an imperceptible watermark by modifying a selected pixel (e.g., +5.0). Only differences above the threshold are detectable in a visual diff map. PixCrypt achieves a 13.1× speedup, reducing time from 63.08s to 4.8s.

Image Segmentation. The encrypted image segmentation performs binary thresholding via a third-degree Taylor approximation of the tanh function to

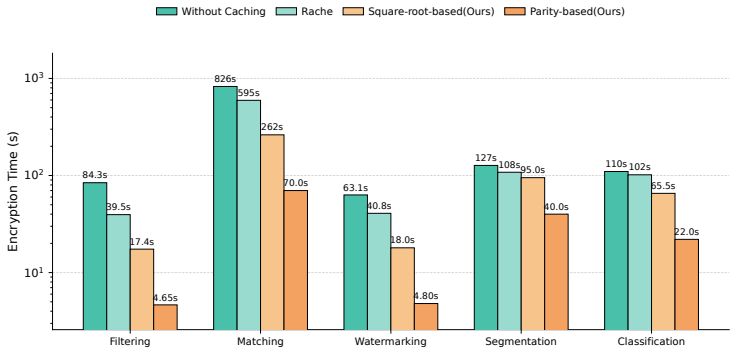

Fig. 4. Encryption time (s) across real-world applications. PixCrypt consistently reduces runtime across all tasks, achieving up to 18× speedup.

replace direct pixel comparisons. The proposed PixCrypt framework reduces encryption time for a 128×128 image by 3×, from 127 s to 40 s.

Binary Classification. We implement binary classification on encrypted feature vectors using SIMD (Single Instruction, Multiple Data) batching. A compact CNN and quantized logistic head enable efficient encrypted inference, where each vector is packed into one ciphertext for linear scoring. This SIMD-aware design parallelizes computation and reduces runtime from 110 s to 22 s with accuracy comparable to plaintext.

4.5 Bootstrapping Frequency Analysis

To empirically validate the noise reduction claim of Section 3.3, we measure the bootstrapping frequency during image segmentation, one of the deepest circuits in our evaluation. For a target value v, the kernel approximates $\tanh(v) \approx v - \frac{v^3}{3}$, which requires two sequential ciphertext–ciphertext multiplications and is therefore sensitive to noise accumulation. We set the bootstrapping threshold to the minimum ciphertext level $\ell_{\min}$ below which a bootstrap is triggered, and evaluate all methods under identical CKKS parameters and the same $\ell_{\min}$.

Let σ_0 be the noise of a freshly constructed ciphertext, thr the noise threshold, and $\Delta_{\text{add}}, \Delta_{\text{mult}}$ the per-operation increments of plaintext–ciphertext and ciphertext–ciphertext operations. The available budget is $B = \mathsf{thr} - \sigma_0$ and the operational depth before a bootstrap is $d = \lfloor B/\Delta \rfloor$. A reconstruction that stays in the additive, plaintext-oriented regime grows noise linearly,

$$\text{noise}(t) = \sigma_0 + \sum_{j=1}^{t} \Delta_{\text{add},j} + O(\log M) = O(\sigma_0 + t\,\Delta_{\text{add}} + \log M)\,, \tag{6}$$

in contrast to the multiplicative accumulation $O(\sigma_0 \cdot \Delta_{\text{mult}}^t)$ of chained ciphertext–ciphertext construction, which yields

$$d_{\text{add}} = O\!\left(\frac{\mathsf{thr}}{\Delta_{\text{add}}}\right) \;\gg\; d_{\text{mult}} = O\big(\log_{\Delta_{\text{mult}}} \mathsf{thr}\big)\,. \tag{7}$$

A larger depth d leaves fewer pixels crossing $\ell_{\min}$ within the fixed segmentation circuit, hence fewer bootstraps. The two cached reconstructions differ in level consumption. Parity-based synthesis is purely additive, $\widetilde{\Pi}(t) = \Pi(2v) \oplus \delta\, \Pi(1) \oplus Z$, consuming no multiplicative level and entering the circuit at the full budget B. Square-root synthesis applies a ciphertext–plaintext multiplication $\Pi(k) \odot \mathrm{pt}(k)$, which multiplies the scales $\Delta_{ct} \cdot \Delta_{pt}$ and requires one rescaling $\ell \to \ell - 1$ to restore the target scale, lowering the budget by one level. This predicts $d_{\mathrm{parity}} > d_{\mathrm{sqrt}}$, matching the measured counts.

Table 6. Normalized bootstrapping cost comparison under a constant per-bootstrap time assumption. We report the number of ciphertexts (pixels) requiring at least one bootstrap, along with the normalized bootstrapping time and total runtime.

Method	#Bootstrap	BS Time (s)	Enc Time (s)	Total (s)	BS %
Without Caching	64	164.463	0.931	165.394	99.4%
Rache	41	105.400	0.063	105.463	99.9%
Sqrt-based (Ours)	34	87.400	0.287	87.687	99.7%
Parity-based (Ours)	27	69.400	0.065	69.465	99.9%

Table 6 reports the number of bootstrapping operations, total bootstrapping time, encryption time, and the runtime fraction attributable to bootstrapping. Without caching, all 64 pixels bootstrap, confirming that independent fresh encryption offers no noise advantage. Rache reduces this to 41, but its ciphertext–ciphertext additions of independently encrypted components keep the initial noise comparable to the baseline. The square-root method reaches 34 via plaintext–ciphertext multiplication, and the level-preserving parity method attains the fewest at 27, a 2.4× reduction over the baseline. These counts follow the trend implied above: the additive, level-preserving reconstruction sustains the largest depth d and thus the fewest bootstraps. Because bootstrapping accounts for over 99% of runtime in this circuit, the reduction in bootstrap count carries almost directly to the total time, which drops from 165.4 s to 69.5 s, a 2.4× end-to-end speedup spanning encryption, homomorphic evaluation, and bootstrapping together, complementing the encryption-time results of Section 4. The same table shows that encryption accounts for only a small fraction of the runtime in a deep circuit (0.065 s out of 69.5 s). Therefore, the encryption speedups reported in Section 4 are most relevant to shallow, encryption-heavy tasks such as filtering and watermarking, whereas deep circuits benefit primarily from fewer bootstrapping operations.

4.6 Scalability to Higher Bit-Depth Domains

Our evaluation uses 8-bit pixels ($M = 255$), but higher-precision imaging (e.g., 12- or 16-bit medical images, M up to 65,535) raises a scalability question. The parity and square-root caches store $\lfloor M/2 \rfloor + 1$ and $\lfloor \sqrt{M} \rfloor + 1$ ciphertexts, scaling as $O(M)$ and $O(\sqrt{M})$, respectively. Table 7 reports storage and precomputation across bit depths, based on ≈ 125 KB per ciphertext and ≈ 4 ms per encryption

Table 7. Cache size and precomputation time of the two caching strategies across plaintext bit depths. Parity scales as $O(M)$ and square-root as $O(\sqrt{M})$.

Bit depth (M)	Method	#CTs	Cache	Precompute
8-bit (255)	Parity $O(M/2)$	128	16 MB	0.51 s
	Sqrt $O(\sqrt{M})$	16	2 MB	0.06 s
12-bit (4095)	Parity	2048	256 MB	8.2 s
	Sqrt	64	8 MB	0.26 s
16-bit (65535)	Parity	32768	$\sim$ 4 GB	$\sim$ 131 s
	Sqrt	256	32 MB	$\sim$ 1.0 s

at $\log N = 12$; its 8-bit results match our measurements. At 16-bit, parity-based caching needs $\sim$4 GB and $\sim$131 s, which is impractical, whereas square-root-based caching needs only 32 MB and $\sim$1 s and thus extends naturally to high precision. Parity can still be used via a digit-wise split of a w-bit value into $\lceil w/8 \rceil$ bytes over a shared 8-bit cache, keeping storage at $O(256)$.

5 Conclusion

This work introduces two orthogonal caching-based strategies to accelerate pixel-level fully homomorphic encryption (FHE): a *square-root-based* method that decomposes values into square and remainder terms, and a *parity-based* method that represents plaintexts as even bases plus parity bits. Both methods avoid ciphertext–ciphertext multiplications and reduce bootstrapping. We show that both methods preserve IND-CPA security under CKKS, BFV, and BGV, and experiments report up to **35**$\times$ speedups per image.

Acknowledgments. This work was supported by the National Science Foundation (NSF) under awards 2409851, 2528534, and 2403603, with additional partial support from NSF awards 1921576 and 2334196.

References

1. Usc-sipi image database website. http://sipi.usc.edu/database/database.php?volume=misc (1981)
2. Lattigo v2.4.0. Online: https://github.com/ldsec/lattigo (Jan 2022), ePFL-LDS
3. Acar, A., Aksu, H., Uluagac, A.S., Conti, M.: A survey on homomorphic encryption schemes: Theory and implementation. ACM Computing Surveys (Csur) **51**(4), 1–35 (2018)
4. Agrawal, R., de Castro, L., Yang, G., Juvekar, C., Yazicigil, R., Chandrakasan, A., Vaikuntanathan, V., Joshi, A.: Fab: An fpga-based accelerator for bootstrappable fully homomorphic encryption. In: 2023 IEEE International Symposium on High-Performance Computer Architecture (HPCA). pp. 882–895. IEEE (2023)
5. Agrawal, R., De Castro, L., Juvekar, C., Chandrakasan, A., Vaikuntanathan, V., Joshi, A.: Mad: Memory-aware design techniques for accelerating fully homomorphic encryption. In: Proceedings of the 56th Annual IEEE/ACM International Symposium on Microarchitecture. pp. 685–697 (2023)

6. Albrecht, M., Chase, M., Chen, H., Ding, J., Goldwasser, S., Gorbunov, S., Halevi, S., Hoffstein, J., Laine, K., Lauter, K., Lokam, S., Micciancio, D., Moody, D., Morrison, T., Sahai, A., Vaikuntanathan, V.: Homomorphic encryption standard. Cryptology ePrint Archive, Paper 2019/939 (2019), https://eprint.iacr.org/2019/939
7. Blatt, M., Gusev, A., Polyakov, Y., Rohloff, K., Vaikuntanathan, V.: Optimized homomorphic encryption solution for secure genome-wide association studies. BMC Medical Genomics **13**, 1–13 (2020)
8. Boemer, F., Cammarota, R., Demmler, D., Schneider, T., Yalame, H.: Mp2ml: A mixed-protocol machine learning framework for private inference. In: Proceedings of the 15th international conference on availability, reliability and security. pp. 1–10 (2020)
9. Brakerski, Z.: Fully homomorphic encryption without modulus switching from classical gapsvp. Annual cryptology conference pp. 868–886 (2012)
10. Brakerski, Z., Gentry, C., Vaikuntanathan, V.: (leveled) fully homomorphic encryption without bootstrapping. ACM Transactions on Computation Theory (TOCT) **6**(3), 1–36 (2014)
11. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. Advances in Cryptology – ASIACRYPT 2017 pp. 409–437 (2017)
12. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: TFHE: Fast fully homomorphic encryption over the torus. Cryptology ePrint Archive, Paper 2018/421 (2018), https://eprint.iacr.org/2018/421
13. Dwork, C.: Differential privacy: A survey of results. In: International conference on theory and applications of models of computation. pp. 1–19. Springer (2008)
14. erkan, u., Thanh, D.N.H., Hieu, L.M., Engínoğlu, S.: An iterative mean filter for image denoising. IEEE Access **7**, 167847–167859 (2019). https://doi.org/10.1109/ACCESS.2019.2953924
15. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive, Paper 2012/144 (2012), https://eprint.iacr.org/2012/144
16. Fridrich, J.: Image watermarking for tamper detection. In: Proceedings 1998 International Conference on Image Processing. ICIP98 (Cat. No. 98CB36269). vol. 2, pp. 404–408. IEEE (1998)
17. Geelen, R., Van Beirendonck, M., Pereira, H.V., Huffman, B., McAuley, T., Selfridge, B., Wagner, D., Dimou, G., Verbauwhede, I., Vercauteren, F., et al.: Basalisc: Programmable asynchronous hardware accelerator for bgv fully homomorphic encryption. arXiv preprint arXiv:2205.14017 (2022)
18. Genise, N., Micciancio, D., Polyakov, Y.: Building an efficient lattice gadget toolkit: Subgaussian sampling and more. In: Ishai, Y., Rijmen, V. (eds.) Advances in Cryptology – EUROCRYPT 2019. pp. 655–684. Springer International Publishing, Cham (2019)
19. Gentry, C.: Fully homomorphic encryption using ideal lattices. Proceedings of the forty-first annual ACM symposium on Theory of computing (2009)
20. Gupta, S., Rosing, T.Š.: Invited: Accelerating fully homomorphic encryption with processing in memory. In: 2021 58th ACM/IEEE Design Automation Conference (DAC). pp. 1335–1338 (2021). https://doi.org/10.1109/DAC18074.2021.9586285
21. Halevi, S., Polyakov, Y., Shoup, V.: An improved rns variant of the bfv homomorphic encryption scheme. Topics in Cryptology – CT-RSA 2019 pp. 83–105 (2019)
22. Jiang, L., Lou, Q., Joshi, N.: Matcha: a fast and energy-efficient accelerator for fully homomorphic encryption over the torus. In: Proceedings of the 59th ACM/IEEE Design Automation Conference. p. 235–240. DAC '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3489517.3530435

23. Jiang, M., Zhu, Y., You, H., Tan, C., Li, Z., Xu, J., Ju, L.: Fhe-cgra: Enable efficient acceleration of fully homomorphic encryption on cgras. In: Proceedings of the 61st ACM/IEEE Design Automation Conference. DAC '24, Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3649329.3656536
24. Kim, S., Kim, J., Kim, M.J., Jung, W., Kim, J., Rhu, M., Ahn, J.H.: Bts: an accelerator for bootstrappable fully homomorphic encryption. In: Proceedings of the 49th Annual International Symposium on Computer Architecture. p. 711–725. ISCA '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3470496.3527415
25. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. International journal of computer vision **60**(2), 91–110 (2004)
26. Mu, J., Han, H., Shi, S., Ye, J., Liu, Z., Liang, S., Li, M., Zhang, M., Bian, S., Hu, X., Li, H., Li, X.: Alchemist: A unified accelerator architecture for cross-scheme fully homomorphic encryption. In: Proceedings of the 61st ACM/IEEE Design Automation Conference. DAC '24, Association for Computing Machinery, New York, NY, USA (2024). https://doi.org/10.1145/3649329.3657331
27. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) Advances in Cryptology — EUROCRYPT '99. pp. 223–238. Springer Berlin Heidelberg, Berlin, Heidelberg (1999)
28. Ren, X., Chen, Z., Gu, Z., Lu, Y., Zhong, R., Lu, W.J., Zhang, J., Zhang, Y., Wu, H., Zheng, X., Liu, H., Chu, T., Hong, C., Wei, C., Niu, D., Xie, Y.: Cham: A customized homomorphic encryption accelerator for fast matrix-vector product. In: 2023 60th ACM/IEEE Design Automation Conference (DAC). pp. 1–6 (2023). https://doi.org/10.1109/DAC56929.2023.10247696
29. Rivest, R.L., Adleman, L., Dertouzos, M.L., et al.: On data banks and privacy homomorphisms. Foundations of secure computation **4**(11), 169–180 (1978)
30. Savvides, S., Khandelwal, D., Eugster, P.: Efficient confidentiality-preserving data analytics over symmetrically encrypted datasets. Proc. VLDB Endow. **13**(8), 1290–1303 (apr 2020). https://doi.org/10.14778/3389133.3389144
31. Timothy Tawose, O., et al.: Toward efficient homomorphic encryption for outsourced databases through parallel caching. Proc. ACM Manag. Data **1**(1) (may 2023). https://doi.org/10.1145/3588920
32. Van Beirendonck, M., D'Anvers, J.P., Turan, F., Verbauwhede, I.: Fpt: A fixed-point accelerator for torus fully homomorphic encryption. In: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. p. 741–755. CCS '23, Association for Computing Machinery, New York, NY, USA (2023). https://doi.org/10.1145/3576915.3623159
33. Yang, H., Shen, S., Dai, W., Zhou, L., Liu, Z., Zhao, Y.: Phantom: A cuda-accelerated word-wise homomorphic encryption library. IEEE Transactions on Dependable and Secure Computing **21**(5), 4895–4906 (2024). https://doi.org/10.1109/TDSC.2024.3363900
34. Yune, S., Lee, H., Putra, A., Cho, H., Manh, C.D., Jeon, J., Kim, J.Y.: Abc-fhe: A resource-efficient accelerator enabling bootstrappable parameters for client-side fully homomorphic encryption. In: 2025 62nd ACM/IEEE Design Automation Conference (DAC). pp. 1–7 (2025). https://doi.org/10.1109/DAC63849.2025.11132592
35. Zhao, C., Zhao, S., Zhao, M., Chen, Z., Gao, C.Z., Li, H., Tan, Y.a.: Secure multi-party computation: theory, practice and applications. Information Sciences **476**, 357–372 (2019)