HELITE-RESNET: A CKKS-FRIENDLY RESIDUAL NETWORK FOR END-TO-END ENCRYPTED FACIAL EXPRESSION RECOGNITION

Anonymous authors
Paper under double-blind review

ABSTRACT

Since facial expression recognition (FER) processes sensitive biometric data, privacy concerns arise. Secure multiparty computation and trusted execution environments are common but require multiparty interaction and trusted hardware, respectively. Fully homomorphic encryption instead enables non-interactive, end-to-end encrypted inference. Unlike exact-integer FHE schemes, CKKS supports approximate arithmetic on real and complex numbers and SIMD packing. However, CKKS-based residual networks still incur substantial computational and memory overheads. Prior work optimizes ciphertext packing, convolution mapping, and polynomial activations, while global costs from stage topology and cross-resolution residual paths remain underexplored. We propose HELite-ResNet, a CKKS-friendly residual network for end-to-end encrypted FER. The proposed framework is the first CKKS-based residual-network framework for encrypted FER. Guided by encrypted execution cost, HELite-ResNet reconfigures ResNet20 stages and proposes an FHE-friendly HELite Transition to reduce homomorphic computation overhead. Knowledge distillation maintains performance without deployment overhead. Compared with the fastest FIDESlib-ResNet20 baseline, HELite-ResNet reduces average encrypted inference latency by 28.6% across CIFAR-10, FERPlus, and RAF-DB, whilst the encrypted accuracy is even slightly higher with the identical CKKS parameters and peak GPU memory usage.

1 INTRODUCTION

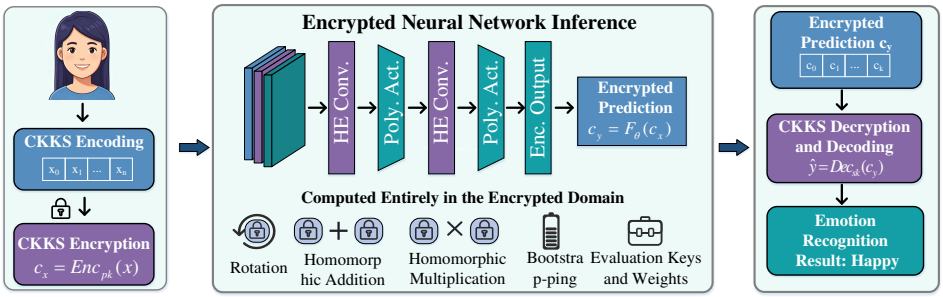
Emotion recognition enables intelligent systems to better perceive user states and improve human–computer interaction, with facial expressions serving as an important source of information (Wen et al., 2023). However, cloud-based facial expression recognition typically requires servers to process facial images in plaintext, potentially exposing sensitive information such as user identity and facial characteristics (Xu et al., 2025). Privacy-preserving inference can rely on secure multiparty computation (SMPC), trusted execution environments (TEEs), or FHE. Optimized SMPC and hybrid protocols still require online interaction and communication (Juvekar et al., 2018; Mishra et al., 2020), while TEEs rely on trusted hardware (Tramèr & Boneh, 2019).

FHE avoids these requirements by enabling non-interactive, end-to-end inference over encrypted inputs, making it attractive for privacy-preserving facial expression recognition. As shown in Figure 1, the client encrypts an image, the server executes the neural network in the encrypted domain and returns an encrypted prediction, and the client finally decrypts the result. Since deep neural network inference involves extensive real-valued computation, and CKKS supports approximate homomorphic computation over encrypted real-valued data, it is well suited to encrypted neural network inference (Cheon et al., 2017; 2019).

Encrypted-execution bottlenecks in residual networks. Residual networks (He et al., 2016) are commonly used for facial expression recognition (Wen et al., 2023), and ResNet20 is a representative FHE benchmark used by Lee et al. (2022a), Lee et al. (2022b), and Rovida & Leporati (2024). We therefore study its encrypted inference for privacy-preserving facial expression recognition. In CKKS, rotations, multiplications, and bootstrapping are costly (Cheon et al., 2019; Bossuat et al., 2021), while parameter counts and FLOPs do not reflect encrypted inference cost (Lou & Jiang, 2021; Ao

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

& Boddeti, 2024). Prior work shows that packing and layout choices affect subsequent encrypted computation (Kim et al., 2024; Ju et al., 2024); cross-resolution residual paths can propagate these effects to later stages, making local choices affect global cost. Residual networks should therefore be designed from the global encrypted execution graph.



(a) Client-side input and encryption (b) Server-side encrypted inference (c) Client-side result decryption

Figure 1: End-to-end encrypted inference for privacy-preserving facial-expression recognition: (a) client-side encryption, (b) server-side encrypted inference, and (c) client-side decryption. HE Conv., Poly. Act., and Enc. Output denote homomorphic convolution, polynomial activation, and encrypted classification output, respectively.

Cost-driven network restructuring. Existing studies optimize packing and convolution mapping (Lee et al., 2022a; Kim & Guyot, 2023; Kim et al., 2024), polynomial activations and FHE-aware adaptation (Park et al., 2022; Ao & Boddeti, 2024), and bootstrapping or execution scheduling (Ju et al., 2024; Cheon et al., 2024); recent work also considers topology–homomorphic-cost co-design (Lou & Jiang, 2021; Ao & Boddeti, 2025). However, the global CKKS cost induced by residual stage topology and cross-resolution paths remains underexplored. We propose HELite-ResNet, a cost-driven lightweight residual network for CKKS encrypted inference. To the best of our knowledge, it is the first CKKS-based residual-network framework for end-to-end facial expression recognition over encrypted facial images. HELite-ResNet restructures ResNet20 stages and proposes an FHE-friendly HELite Transition to reduce expensive homomorphic computation costs. Meanwhile, knowledge distillation preserves model accuracy without additional encrypted computation.

End-to-end encrypted inference results. Under identical CKKS parameters, runtime, and hardware settings, HELite-ResNet maintains classification performance and high plaintext–ciphertext prediction agreement while reducing the average server-side encrypted inference time from 49.29 s to 35.20 s, corresponding to a 28.6% reduction. The numbers of polynomial activations, multiplications, ciphertext rotations, and bootstrapping operations are reduced by 21.1%, 24.6%, 30.3%, and 37.0%, respectively. The source code of HELite-ResNet is available at: `https://anonymous.4open.science/r/code-submission-2252/`.

Main contributions.

(1) To the best of our knowledge, we propose the first CKKS-based residual-network framework for end-to-end facial expression recognition over encrypted facial images, with facial images remaining encrypted throughout server-side neural-network inference.
(2) We propose the lightweight HELite-ResNet, which optimizes the stage structure of ResNet20 according to encrypted execution cost and uses training-time knowledge distillation to preserve classification performance.
(3) We design an FHE-friendly HELite Transition that performs spatial downsampling and slot compaction before channel expansion and adopts parameter-free DownsampleA, thereby reducing ciphertext convolutions, rotations, and slot rearrangements. It also optimizes the placement of bootstrapping operations, reducing the number of bootstrapping operations per Transition from 6 to 3.
(4) We evaluate the proposed method on CIFAR-10, FERPlus, and RAF-DB, demonstrating that HELite-ResNet reduces average encrypted inference latency by 28.6% compared with the fastest FIDESlib-ResNet20 baseline, while achieving even slightly higher encrypted accuracy under identical CKKS parameters and peak GPU memory usage.

2 Background and Related Work

2.1 CKKS Operations and Residual-Network Mapping

CKKS is a homomorphic encryption scheme that supports approximate arithmetic over real numbers (Cheon et al., 2017; 2019) and uses SIMD slots to process multiple values in parallel. It supports addition, ciphertext–plaintext multiplication, ciphertext–ciphertext multiplication, and slot rotations, but cannot directly evaluate non-polynomial operations such as ReLU. Slot rotations require key switching, while ciphertext–ciphertext multiplication requires relinearization and rescaling and consumes modulus levels; these operations are therefore costly. When the remaining levels are insufficient, bootstrapping refreshes the ciphertext to continue computation and usually incurs the highest cost (Bossuat et al., 2021). When performing residual-network inference on CKKS-encrypted images, convolution, activation, residual connections, and downsampling must all be expressed as ciphertext operations constrained by slot layouts, scales, and modulus levels. Convolution aligns spatial neighborhoods through slot rotations, followed by multiplication with plaintext weights and accumulation; batch normalization can be fused into an adjacent convolution before deployment, while ReLU and other nonlinear functions are approximated with polynomials. Residual addition requires the main and shortcut branches to have compatible output shapes, slot layouts, scales, and remaining modulus levels (Lee et al., 2022a; Kim & Guyot, 2023; Kim et al., 2024; Rovida & Leporati, 2024). Let $[f]$ and $[s]$ denote the ciphertext outputs of the two branches. Then,

$$[y] = [f] \oplus [s], \qquad \mathcal{P}_f = \mathcal{P}_s, \quad \ell_f = \ell_s, \quad \Delta_f \approx \Delta_s, \tag{1}$$

where $\mathcal{P}$, ℓ, and Δ denote the slot layout, modulus level, and scale, respectively.

2.2 Related Work and Research Gap

CryptoNets and LoLa reduce the computational and memory overheads of encrypted inference through low-degree polynomials and inter-layer representation optimization, respectively (Gilad-Bachrach et al., 2016; Brutzkus et al., 2019). GAZELLE and Delphi combine homomorphic encryption with secure multiparty computation to handle nonlinear layers, but require online interaction (Juvekar et al., 2018; Mishra et al., 2020). For non-interactive CKKS inference, MPCNN, optimized FHE CNN mappings, HyPHEN, and recent CKKS CNN systems mainly improve ciphertext packing and convolution mappings (Lee et al., 2022a; Kim & Guyot, 2023; Kim et al., 2024; Park et al., 2023). AESPA and AutoFHE focus on polynomial activations and FHE-aware network adaptation (Park et al., 2022; Ao & Boddeti, 2024), while NeuJeans, efficient CKKS bootstrapping, and DaCapo optimize bootstrapping and execution scheduling (Ju et al., 2024; Bossuat et al., 2021; Cheon et al., 2024). HEMET and CryptoFace show the need to co-design network topology with homomorphic cost (Lou & Jiang, 2021; Ao & Boddeti, 2025). Prior work explores low-memory-footprint CKKS-based ResNet20 inference (Rovida & Leporati, 2024), while FIDESlib (Agulló-Domingo et al., 2025) provides GPU-accelerated CKKS implementations for efficient homomorphic computation. However, the structural costs of residual stages and cross-resolution paths remain underexplored. How to restructure these components to reduce end-to-end CKKS inference time without sacrificing model utility therefore remains an open problem.

2.3 Design Objectives and Threat Model

Motivated by this gap, HELite-ResNet targets input-private cloud-based facial expression recognition, aiming to reduce end-to-end CKKS inference time while retaining plaintext performance comparable to ResNet20 and plaintext–ciphertext prediction agreement. We assume an honest-but-curious server: the client retains the secret key, while the server receives the encrypted input, evaluation keys, and plaintext model parameters. The server follows the inference protocol but may inspect the information available to it. The input remains encrypted throughout server-side inference.

3 HELITE-RESNET

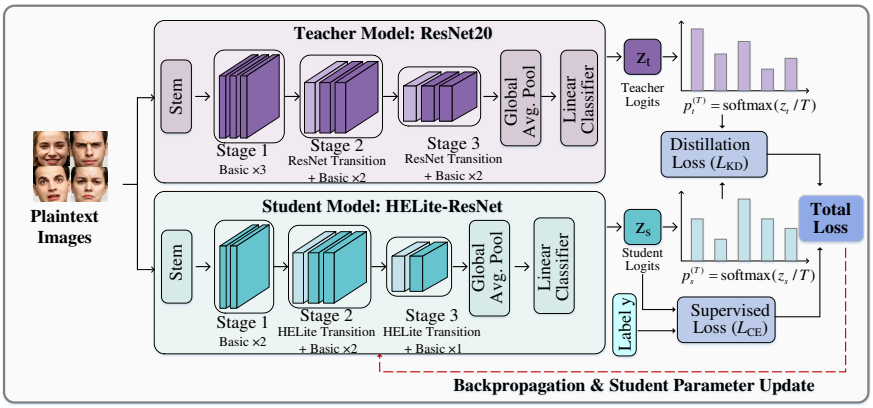
3.1 HELITE-RESNET FRAMEWORK

To address these objectives, we present HELite-ResNet, a lightweight residual network for CKKS encrypted inference. Figure 2 outlines the overall framework, which combines a compact residual topology to reduce ciphertext computation, a lightweight cross-resolution Transition to lower stage-transition overhead, and knowledge distillation during plaintext training to preserve classification performance.

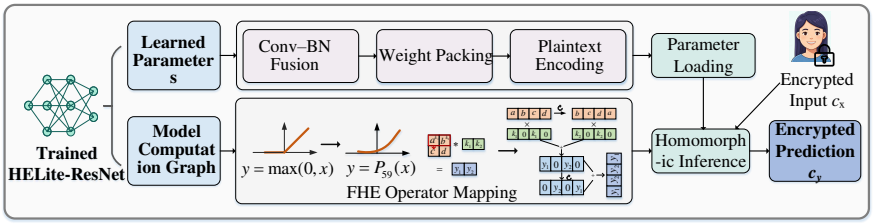
HELite-ResNet builds on ResNet20 (He et al., 2016) and changes the BasicBlock allocation across the three stages from $[3, 3, 3]$ to $[2, 3, 2]$. It selectively removes the terminal standard block in Layer1 and Layer3 while retaining all three BasicBlocks in Layer2. Layer1 has the largest spatial resolution, whereas Layer3 has the widest channel dimension. Shortening repeated residual paths in these stages therefore reduces high-cost ciphertext computation.

We further design the lightweight HELite Transition to connect adjacent stages. Unlike conventional transitions that expand channels before encrypted layout transformation, HELite Transition performs spatial compaction before channel expansion and relocates bootstrapping operations, reducing ciphertext overhead and bootstrapping frequency during cross-resolution conversion.

To compensate for the classification-performance loss induced by the lightweight architecture, we introduce logit-based knowledge distillation during offline plaintext training. A frozen ResNet20 teacher supervises HELite-ResNet. After training, the system compiles only HELite-ResNet into a CKKS ciphertext computation graph for deployment. The teacher network and distillation loss remain outside online inference and therefore incur no additional ciphertext computation.



(a) Offline plaintext distillation training



(b) Online encrypted inference

Figure 2: Plaintext knowledge-distillation training and encrypted inference pipeline of HELite-ResNet. (a) Offline training uses a frozen ResNet20 teacher to supervise the trainable HELite-ResNet student. (b) Online encrypted inference uses only the trained student after FHE operator mapping and parameter preparation.

3.2 Homomorphic Convolution and Polynomial Activation

Packed convolution is performed over CKKS slots. Let ct denote an input ciphertext with logical shape $C \times H \times W$, where each channel block contains $S = HW$ spatial slots. For a 3×3 convolution, let the nine rotation offsets be $(t_0, \ldots, t_8) = (-W-1, -W, -W+1, -1, 0, 1, W-1, W, W+1)$, and let $R_m = \mathrm{Rot}(ct, t_m)$ for $m = 0, \ldots, 8$. For each channel diagonal d, the spatial neighborhoods are multiplied slot-wise by the corresponding plaintext-encoded weights and accumulated:

$$D_d = \sum_{m=0}^{8} R_m \odot P_{d,m}, \qquad d = 0, \ldots, C-1. \tag{2}$$

Here, $\odot$ denotes slot-wise ciphertext-plaintext multiplication, and C is the channel count. During model export, the convolution and BN parameters are fused, and the resulting weights and layout masks are encoded as plaintext weights $P_{d,m}$, while the plaintext bias P_b is also constructed. The diagonal outputs are then aligned to the channel layout and accumulated to form the convolution output:

$$ct_{\mathrm{out}} = \sum_{d=0}^{C-1} \mathrm{Rot}(D_d, -dS) + P_b. \tag{3}$$

Here, ct_{out} denotes the convolution output ciphertext, and S denotes the number of slots per channel block.

Since CKKS cannot directly evaluate comparison-based ReLU, activations are scaled to $[-1, 1]$, and the nonlinearity is approximated by a degree-59 Chebyshev polynomial (Rovida & Leporati, 2024):

$$P_{59}(x) = \sum_{j=0}^{59} a_j T_j(x), \qquad x \in [-1, 1]. \tag{4}$$

Here, T_j is the j-th Chebyshev basis function, and a_j is the corresponding coefficient. Bootstrapping is performed before polynomial activation.

3.3 Proposed HELite Transition

Original ResNet transitions (Rovida & Leporati, 2024; Agulló-Domingo et al., 2025) are inefficient under CKKS because they perform channel expansion before slot compaction and rely on convolution-based transformations, introducing additional ciphertext computations.

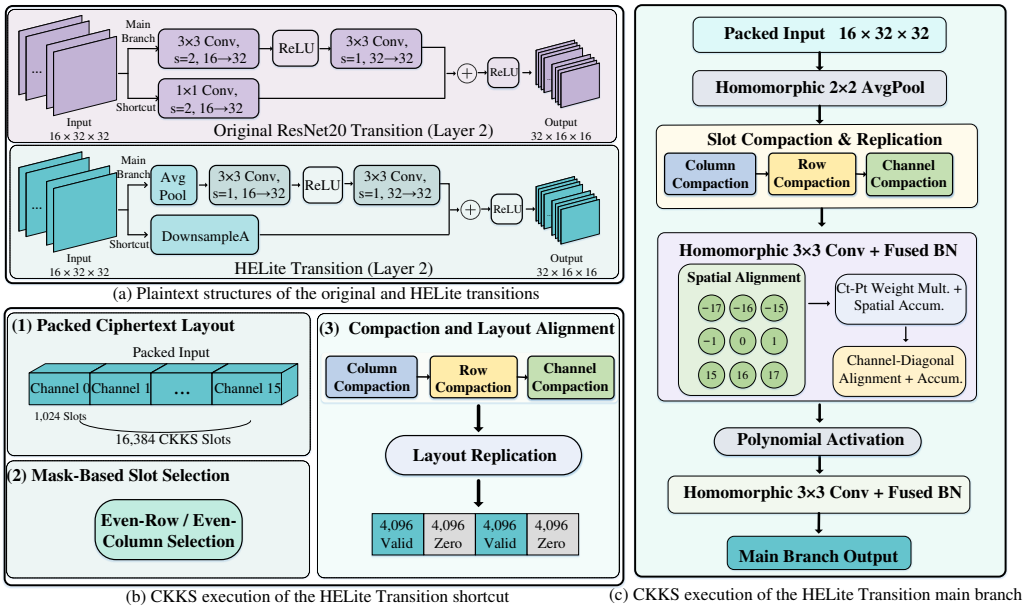
To address these issues, HELite Transition adopts a compact-before-expand strategy. The overall structure is illustrated in Fig. 3. It first performs average pooling and slot compaction on the input ciphertext ct_{in}, whose logical shape is $C_{\mathrm{in}} \times H \times W$, and then expands the channels:

$$ct_{\mathrm{pool}} = \big(ct_{\mathrm{in}} + \mathrm{Rot}(ct_{\mathrm{in}}, 1) + \mathrm{Rot}(ct_{\mathrm{in}}, W) + \mathrm{Rot}(ct_{\mathrm{in}}, W+1)\big) \odot M_{\mathrm{avg}}. \tag{5}$$

Here, C_{in}, H, and W denote the input channel count, feature height, and feature width, respectively; $\mathrm{Rot}(ct, s)$ denotes a cyclic rotation of ciphertext ct by s slots; M_{avg} is the plaintext average-pooling mask that applies the $1/4$ scaling; and $\odot$ denotes slot-wise multiplication. Subsequently,

$$ct_{\mathrm{cmp}} = \Pi_{\mathrm{ch}}(\Pi_{\mathrm{row}}(\Pi_{\mathrm{col}}(ct_{\mathrm{pool}}))). \tag{6}$$

Π_{col}, Π_{row}, and Π_{ch} denote column, row, and channel compaction, respectively, implemented through rotations, additions, and plaintext masks. ct_{cmp} is replicated to form the convolution layout ct_{pack}.



(a) Plaintext structures of the original and HELite transitions

(b) CKKS execution of the HELite Transition shortcut

(c) CKKS execution of the HELite Transition main branch

Figure 3: Structural and ciphertext-level comparison of the original and HELite transitions: (a) plaintext structures, (b) CKKS execution of the HELite shortcut, and (c) CKKS execution of the HELite main branch.

The main branch takes ct_{pack} as input and performs two homomorphic 3×3 convolutions; the first convolution is followed by polynomial activation. The shortcut adopts the parameter-free DownsampleA operation for spatial downsampling and channel alignment before residual addition. Compared with the original transition, HELite reduces cross-channel rotations and alignment operations by compacting before channel expansion, while removing the 1×1 ciphertext convolution and subsequent slot rearrangement through the parameter-free shortcut.

In addition, HELite Transition redesigns the bootstrapping placement. The original Transition requires bootstrapping after channel expansion, downsampling, and residual addition, whereas HELite performs bootstrapping once before duplicating the compacted ciphertext into the two branches, reducing bootstrapping operations from six to three. The detailed ciphertext execution flow and bootstrapping schedules are provided in Appendix C.

3.4 Knowledge Distillation Training

To mitigate the classification-performance loss caused by the lightweight architecture, we apply logit distillation (Hinton et al., 2015) during plaintext training. The full ResNet20 serves as the teacher, and HELite-ResNet serves as the student. Let z_t and z_s denote the unnormalized classifier outputs (logits) of the teacher and student, respectively, and let y denote the ground-truth label. The training objective is

$$\mathcal{L} = (1-\alpha)\mathcal{L}_{\text{CE}} + \alpha T_{\text{KD}}^2 \mathcal{L}_{\text{KD}}, \qquad \mathcal{L}_{\text{KD}} = D_{\text{KL}}\left(p_t^{(T_{\text{KD}})} \,\middle\|\, p_s^{(T_{\text{KD}})}\right). \tag{7}$$

Here, $\mathcal{L}_{\text{CE}}$ denotes the cross-entropy loss between z_s and y, and D_{KL} denotes the Kullback-Leibler (KL) divergence. The distributions $p_t^{(T_{\text{KD}})} = \text{softmax}(z_t/T_{\text{KD}})$ and $p_s^{(T_{\text{KD}})} = \text{softmax}(z_s/T_{\text{KD}})$ are the temperature-scaled softmax distributions of the teacher and student, respectively. We set $T_{\text{KD}} = 4$ and $\alpha = 0.7$, where T_{KD} is the temperature and α is the distillation-loss weight.

During distillation, the teacher remains in evaluation mode with frozen parameters, and gradients update only the student. At deployment, we map only HELite-ResNet to the CKKS encrypted computation graph. Relative to an undistilled student with the same architecture, knowledge distillation leaves the encrypted graph and homomorphic operation count unchanged.

4 EVALUATION

4.1 EXPERIMENTAL SETUP AND EVALUATION PROTOCOL

We evaluate general image classification on CIFAR-10 (Krizhevsky, 2009) and facial expression recognition on FERPlus (Barsoum et al., 2016) and RAF-DB (Li et al., 2017). For CIFAR-10 and RAF-DB, we report test-set Top-1 accuracy. For FERPlus, we report best-val and the corresponding test@best-val. We evaluate encrypted correctness using 1,000 plaintext-ciphertext sample pairs per dataset. Latency measurements use 100 valid ciphertext samples and include server-side encrypted forward computation and runtime layer-wise weight loading.

The ResNet20 baseline adopts the CKKS packing mapping from the FIDESlib GPU encrypted-inference example (Agulló-Domingo et al., 2025), following the implementation of Rovida & Leporati (2024). Both models use the same data preprocessing, FIDESlib backend, degree-59 Chebyshev activation approximation, RTX 4090 GPU, and CKKS configuration, with ring dimension 2^{16}, 16,384 packed slots, and 128-bit classical security. We train all plaintext models for 200 epochs using a batch size of 128, an initial learning rate of 0.1, momentum of 0.9, and weight decay of 5×10^{-4}.

4.2 PLAINTEXT UTILITY AND ENCRYPTED PERFORMANCE

This experiment compares ResNet20 and HELite-ResNet under plaintext and encrypted execution on CIFAR-10, FERPlus, and RAF-DB. The comparison assesses whether HELite-ResNet can reduce server-side encrypted inference time while preserving plaintext accuracy and plaintext–ciphertext prediction agreement.

Table 1: Plaintext and encrypted performance of ResNet20 and HELite-ResNet.

Dataset	Model	Full PT (%)	Paired PT (%)	Enc. (%)	Agree. (%)	Time (s)	Speedup
CIFAR-10	ResNet20	92.80	93.30	93.10	96.9	49.22 ± 1.25	1.00×
	HELite-ResNet	92.95 (+0.15)	93.70 (+0.40)	93.50 (+0.40)	98.1 (+1.2)	35.60 ± 1.35 (-27.7%)	1.38×
FERPlus	ResNet20	86.28 / 84.94	84.10	84.20	95.5	49.40 ± 1.23	1.00×
	HELite-ResNet	86.62 / 84.84 (+0.34 / -0.10)	84.30 (+0.20)	84.50 (+0.30)	96.2 (+0.7)	34.90 ± 1.17 (-29.3%)	1.42×
RAF-DB	ResNet20	83.90	84.40	84.10	96.9	49.25 ± 1.25	1.00×
	HELite-ResNet	84.45 (+0.55)	84.60 (+0.20)	84.30 (+0.20)	97.3 (+0.4)	35.11 ± 1.22 (-28.7%)	1.40×

Full PT denotes plaintext accuracy on the complete dataset, and Paired PT denotes plaintext accuracy on samples used for encrypted evaluation. Enc. denotes encrypted accuracy, and Agree. denotes plaintext–ciphertext prediction agreement. The Time column reports the mean ± sample standard deviation.

Table 1 shows that HELite-ResNet improves Full PT accuracy by 0.15 and 0.55 percentage points on CIFAR-10 and RAF-DB, respectively. On FERPlus, HELite-ResNet maintains comparable classification performance to ResNet20. Its plaintext–ciphertext prediction agreement reaches 96.2% to 98.1%, exceeding ResNet20 by 0.4 to 1.2 percentage points. HELite-ResNet also reduces server-side encrypted inference time by 28.6% on average. These results show that HELite-ResNet reduces encrypted inference time while preserving plaintext accuracy and plaintext–ciphertext prediction agreement.

4.3 COMPARISON WITH EXISTING ENCRYPTED INFERENCE METHODS ON CIFAR-10

To further examine the overall performance of HELite-ResNet in the context of existing encrypted inference studies on CIFAR-10, Table 2 summarizes the accuracy, inference time, and memory information reported for several related methods. It also presents measurements of FIDESlib-ResNet20 and HELite-ResNet under the same experimental configuration.

Overall, the reported inference times of existing CPU-based methods range from hundreds to thousands of seconds, whereas the two GPU implementations operate on the order of tens of seconds.

HELite-ResNet achieves an encrypted accuracy of 93.50% with an inference time of 35.60 s, demonstrating a favorable accuracy–efficiency balance and offering a feasible model design for accurate and efficient encrypted inference.

Table 2: Performance comparison of encrypted inference methods on CIFAR-10.

Reference	Scheme	Enc. Acc. (%)	Time (s)	Platform	Memory (GB)
(Lee et al., 2022b)	CKKS	92.43±2.65	10,602	Xeon Platinum 8280, 64 threads	512*
(Lee et al., 2022a)	CKKS	91.31	2,271	AMD TR Pro 3995WX, 1 thread	512*
(Benamira et al., 2025)	TFHE	74.10	522	Intel i7-8650U, 4 threads	0.0475
(Kim & Guyot, 2023)	CKKS	92.04	255	AMD EPYC 7402P, 1 thread	100
(Rovida & Leporati, 2024)	CKKS	91.53	260	Apple M1 Pro, 1 thread	15.1
(Agulló-Domingo et al., 2025)	CKKS	93.10	49.22	NVIDIA RTX 4090	22.645
Ours	CKKS	93.50	35.60	NVIDIA RTX 4090	22.645

The reported results are collected from the original papers. Prior summaries (Rovida & Leporati, 2024) are used for cross-checking. TT-TFHE uses hybrid inference, where the client performs part of the computation in plaintext; the other compared methods use end-to-end encrypted inference. FIDESlib (Agulló-Domingo et al., 2025) provides a GPU example following the ResNet20 encrypted-inference design in (Rovida & Leporati, 2024). * denotes machine RAM capacity; the remaining CPU entries report RAM, whereas the RTX 4090 entries report GPU memory.

4.4 Structural Sources of Encrypted Inference Acceleration

To identify the sources of HELite-ResNet's server-side encrypted inference acceleration, we measure stage, Transition, and full-graph execution costs.

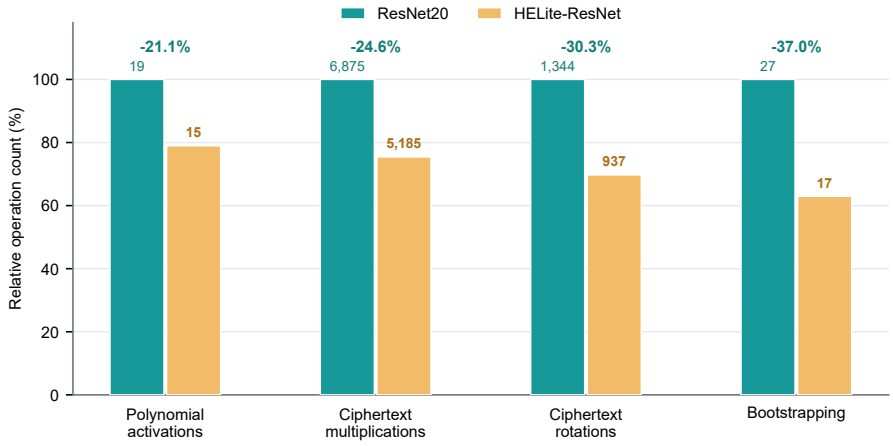


Figure 4: Static homomorphic operation counts for the complete encrypted forward graph with degree-59 polynomial activations. Raw counts appear above the bars; ResNet20 is normalized to 100% per category. Total multiplications include ciphertext–plaintext and ciphertext–ciphertext multiplications or squarings.

Layer1 and Layer3 latency decreases by 34.1% and 38.5%, respectively, making them the primary sources of the acceleration. Layer2 retains all BasicBlocks, yet redesigning its entry Transition lowers whole-stage latency by 15.7%. The shortcut branches of the Transitions entering Layer2 and Layer3 reduce latency by 91.7% and 97.0%, respectively. DownsampleA converts resolution through slot selection and channel alignment, removing encrypted convolution and slot rearrangement from projection shortcuts.

As shown in Figure 4, HELite-ResNet reduces polynomial activations, total multiplications, ciphertext rotations, and bootstrapping operations by 21.1%, 24.6%, 30.3%, and 37.0%, respectively. Total multiplications include ciphertext–plaintext and ciphertext–ciphertext multiplications or squarings. These reductions help explain the measured 28.6% decrease in server-side encrypted inference time, while shorter residual paths and optimized cross-resolution transitions improve encrypted inference efficiency.

4.5 Ablation Study of the Lightweight Architecture and Knowledge Distillation

To evaluate the impact of the lightweight design on plaintext classification performance and examine the respective contributions of the structural modifications and knowledge distillation, we compared five model configurations under the same training conditions, as shown in Table 3.

Table 3: Configuration and plaintext classification accuracy (%) of HELite-ResNet ablations.

Variant	Blocks	Transition	KD	CIFAR-10	FER val	FER test	RAF-DB
ResNet20	No	No	No	92.80	86.28	84.94	83.90
Blocks only	Yes	No	No	91.84 (-0.96)	85.81 (-0.47)	84.94 (0.00)	82.76 (-1.14)
Transition only	No	Yes	No	93.17 (+0.37)	86.31 (+0.03)	84.65 (-0.29)	83.47 (-0.42)
Without KD	Yes	Yes	No	91.64 (-1.16)	86.28 (0.00)	84.87 (-0.06)	83.05 (-0.85)
HELite-ResNet	Yes	Yes	Yes	92.95 (+0.15)	86.62 (+0.34)	84.84 (-0.10)	84.45 (+0.55)

Note: Blocks, Transition, and KD denote residual-block pruning, Transition redesign, and knowledge distillation, respectively; Yes/No indicates whether each component is enabled. Without KD applies both structural modifications without distillation. Parenthesized values are changes relative to ResNet20 in percentage points. FERPlus metrics follow the definition in Section 4.1.

Pruning residual blocks alone reduces accuracy on CIFAR-10 and RAF-DB, whereas Transition redesign has little effect on performance. Applying both structural modifications without distillation decreases accuracy on the two datasets by 1.16 and 0.85 percentage points, respectively. Adding KD improves the complete model over Without KD by 1.31 and 1.40 percentage points, and over ResNet20 by 0.15 and 0.55 percentage points. On FERPlus, its test@best-val is only 0.10 percentage points below that of ResNet20.

These results show that Transition redesign largely preserves classification performance, while knowledge distillation effectively recovers the accuracy loss caused by block pruning. The lightweight model therefore reduces encrypted inference cost while retaining strong plaintext classification performance.

5 Conclusion

This work addresses the privacy requirements of cloud-based facial expression recognition by developing a CKKS-based end-to-end encrypted inference framework. To reduce the high homomorphic execution cost of ResNet20, we propose HELite-ResNet, a lightweight residual network that restructures the stage topology from the perspective of the global encrypted execution graph and proposes an FHE-friendly HELite Transition to optimize cross-resolution encrypted execution paths, while incorporating knowledge distillation to preserve classification performance. Experimental results show that, under identical CKKS parameters and peak GPU memory usage, HELite-ResNet reduces average encrypted inference latency by 28.6% compared with the fastest FIDESlib-ResNet20 baseline while achieving even slightly higher encrypted accuracy. These results demonstrate the effectiveness of architecture restructuring tailored to homomorphic execution characteristics for improving FHE inference efficiency. Future work will further explore the co-design of network architectures and encrypted execution mechanisms to improve the efficiency and scalability of FHE inference.

AI USE STATEMENT

Generative AI tools assisted with language editing, structural organization, and LaTeX preparation. The authors checked all AI-assisted text against the experimental records, source code, and cited literature. The authors take responsibility for the final manuscript, including all claims and artifacts produced with AI assistance.

REFERENCES

Carlos Agulló-Domingo, Óscar Vera-López, Seyda Guzelhan, Lohit Daksha, Aymane El Jerari, Kaustubh Shivdikar, Rashmi Agrawal, David Kaeli, Ajay Joshi, and José L. Abellán. FIDESlib: A fully-fledged open-source FHE library for efficient CKKS on GPUs. In *2025 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 1–3. IEEE, 2025. doi: 10.1109/ISPASS64960.2025.00045. URL https://doi.org/10.1109/ISPASS64960.2025.00045.

Wei Ao and Vishnu Naresh Boddeti. AutoFHE: Automated adaption of CNNs for efficient evaluation over FHE. In *33rd USENIX Security Symposium (USENIX Security 24)*, pp. 2173–2190. USENIX Association, 2024. URL https://www.usenix.org/conference/usenixsecurity24/presentation/ao.

Wei Ao and Vishnu Naresh Boddeti. CryptoFace: End-to-end encrypted face recognition. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 19197–19206, 2025. URL https://openaccess.thecvf.com/content/CVPR2025/html/Ao_CryptoFace_End-to-End_Encrypted_Face_Recognition_CVPR_2025_paper.html.

Emad Barsoum, Cha Zhang, Cristian Canton-Ferrer, and Zhengyou Zhang. Training deep networks for facial expression recognition with crowd-sourced label distribution. In *Proceedings of the 18th ACM International Conference on Multimodal Interaction*, pp. 279–283. Association for Computing Machinery, 2016. doi: 10.1145/2993148.2993165. URL https://doi.org/10.1145/2993148.2993165.

Adrien Benamira, Tristan Guérand, Thomas Peyrin, and Sayandeep Saha. TT-TFHE: a torus fully homomorphic encryption-friendly neural network architecture. *Transactions on Machine Learning Research*, 2025. URL https://openreview.net/forum?id=tV4ynvae6W.

Jean-Philippe Bossuat, Christian Mouchet, Juan Troncoso-Pastoriza, and Jean-Pierre Hubaux. Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In *Advances in Cryptology – EUROCRYPT 2021*, volume 12696 of *Lecture Notes in Computer Science*, pp. 587–617. Springer, 2021. doi: 10.1007/978-3-030-77870-5_21. URL https://doi.org/10.1007/978-3-030-77870-5_21.

Alon Brutzkus, Ran Gilad-Bachrach, and Oren Elisha. Low latency privacy preserving inference. In *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 812–821. PMLR, 2019. URL https://proceedings.mlr.press/v97/brutzkus19a.html.

Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology – ASIACRYPT 2017*, volume 10624 of *Lecture Notes in Computer Science*, pp. 409–437. Springer, 2017. doi: 10.1007/978-3-319-70694-8_15. URL https://doi.org/10.1007/978-3-319-70694-8_15.

Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A full RNS variant of approximate homomorphic encryption. In *Selected Areas in Cryptography – SAC 2018*, volume 11349 of *Lecture Notes in Computer Science*, pp. 347–368. Springer, 2019. doi: 10.1007/978-3-030-10970-7_16. URL https://doi.org/10.1007/978-3-030-10970-7_16.

Seonyoung Cheon, Yongwoo Lee, Dongkwan Kim, Ju Min Lee, Sunchul Jung, Taekyung Kim, Dongyoon Lee, and Hanjun Kim. DaCapo: Automatic bootstrapping management for efficient fully homomorphic encryption. In *33rd USENIX Security Symposium (USENIX Security 24)*, pp. 6993–7010. USENIX Association, 2024. URL https://www.usenix.org/conference/usenixsecurity24/presentation/cheon.

Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. CryptoNets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of the 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pp. 201–210. PMLR, 2016. URL https://proceedings.mlr.press/v48/gilad-bachrach16.html.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 770–778, 2016. doi: 10.1109/CVPR.2016.90. URL https://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015. doi: 10.48550/arXiv.1503.02531. URL https://arxiv.org/abs/1503.02531.

Jae Hyung Ju, Jaiyoung Park, Jongmin Kim, Minsik Kang, Donghwan Kim, Jung Hee Cheon, and Jung Ho Ahn. NeuJeans: Private neural network inference with joint optimization of convolution and FHE bootstrapping. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security*, pp. 4361–4375. Association for Computing Machinery, 2024. doi: 10.1145/3658644.3690375. URL https://doi.org/10.1145/3658644.3690375.

Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. GAZELLE: A low latency framework for secure neural network inference. In *27th USENIX Security Symposium (USENIX Security 18)*, pp. 1651–1669. USENIX Association, 2018. URL https://www.usenix.org/conference/usenixsecurity18/presentation/juvekar.

Donghwan Kim, Jaiyoung Park, Jongmin Kim, Sangpyo Kim, and Jung Ho Ahn. HyPHEN: A hybrid packing method and its optimizations for homomorphic encryption-based neural networks. *IEEE Access*, 12:3024–3038, 2024. doi: 10.1109/ACCESS.2023.3348170. URL https://doi.org/10.1109/ACCESS.2023.3348170.

Dongwoo Kim and Cyril Guyot. Optimized privacy-preserving CNN inference with fully homomorphic encryption. *IEEE Transactions on Information Forensics and Security*, 18:2175–2187, 2023. doi: 10.1109/TIFS.2023.3263631. URL https://doi.org/10.1109/TIFS.2023.3263631.

Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, University of Toronto, 2009. URL https://www.cs.toronto.edu/~kriz/learning-features-2009-TR.pdf.

Eunsang Lee, Joon-Woo Lee, Junghyun Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and Woosuk Choi. Low-complexity deep convolutional neural networks on fully homomorphic encryption using multiplexed parallel convolutions. In *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pp. 12403–12422. PMLR, 2022a. URL https://proceedings.mlr.press/v162/lee22e.html.

Joon-Woo Lee, Hyungchul Kang, Yongwoo Lee, Woosuk Choi, Jieun Eom, Maxim Deryabin, Eunsang Lee, Junghyun Lee, Donghoon Yoo, Young-Sik Kim, and Jong-Seon No. Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. *IEEE Access*, 10:30039–30054, 2022b. doi: 10.1109/ACCESS.2022.3159694. URL https://doi.org/10.1109/ACCESS.2022.3159694.

Shan Li, Weihong Deng, and JunPing Du. Reliable crowdsourcing and deep locality-preserving learning for expression recognition in the wild. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2852–2861, 2017. doi: 10.1109/CVPR.2017.277. URL https://openaccess.thecvf.com/content_cvpr_2017/html/Li_Reliable_Crowdsourcing_and_CVPR_2017_paper.html.

Qian Lou and Lei Jiang. HEMET: A homomorphic-encryption-friendly privacy-preserving mobile neural network architecture. In *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 7102–7110. PMLR, 2021. URL https://proceedings.mlr.press/v139/lou21a.html.

Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: A cryptographic inference service for neural networks. In *29th USENIX Security Symposium (USENIX Security 20)*, pp. 2505–2522. USENIX Association, 2020. URL https://www.usenix.org/conference/usenixsecurity20/presentation/mishra.

Jaiyoung Park, Michael Jaemin Kim, Wonkyung Jung, and Jung Ho Ahn. AESPA: Accuracy preserving low-degree polynomial activation for fast private inference. *arXiv preprint arXiv:2201.06699*, 2022. doi: 10.48550/arXiv.2201.06699. URL https://arxiv.org/abs/2201.06699.

Jaiyoung Park, Donghwan Kim, Jongmin Kim, Sangpyo Kim, Wonkyung Jung, Jung Hee Cheon, and Jung Ho Ahn. Toward practical privacy-preserving convolutional neural networks exploiting fully homomorphic encryption. *arXiv preprint arXiv:2310.16530*, 2023. doi: 10.48550/arXiv.2310.16530. URL https://arxiv.org/abs/2310.16530.

Lorenzo Rovida and Alberto Leporati. Encrypted image classification with low memory footprint using fully homomorphic encryption. *International Journal of Neural Systems*, 34(5):2450025, 2024. doi: 10.1142/S0129065724500254. URL https://doi.org/10.1142/S0129065724500254.

Florian Tramèr and Dan Boneh. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. In *International Conference on Learning Representations (ICLR)*. OpenReview.net, 2019. URL https://openreview.net/forum?id=rJVorjCcKQ.

Zhengyao Wen, Wenzhong Lin, Tao Wang, and Ge Xu. Distract your attention: Multi-head cross attention network for facial expression recognition. *Biomimetics*, 8(2):199, 2023. doi: 10.3390/biomimetics8020199. URL https://doi.org/10.3390/biomimetics8020199.

Feng Xu, David Ahmedt-Aristizabal, Lars Petersson, Dadong Wang, and Xun Li. Facial expression recognition with controlled privacy preservation and feature compensation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, pp. 2611–2621, 2025. doi: 10.1109/WACV61041.2025.00259. URL https://openaccess.thecvf.com/content/WACV2025/html/Xu_Facial_Expression_Recognition_with_Controlled_Privacy_Preservation_and_Feature_Compensation_WACV_2025_paper.html.

APPENDIX A. HELITE-RESNET ARCHITECTURE AND CIPHERTEXT SLOT LAYOUT

A.1 OVERALL NETWORK ARCHITECTURE

HELite-ResNet is built upon ResNet20 and retains its three-stage residual feature extraction framework while changing the stage configuration from the original $[3, 3, 3]$ to $[2, 3, 2]$. Specifically, Layer1 contains two standard BasicBlocks; Layer2 contains one HELite Transition followed by two standard BasicBlocks; and Layer3 contains one HELite Transition followed by one standard BasicBlock. HELite Transition is used at cross-resolution stages to perform spatial downsampling and channel expansion, whereas the remaining BasicBlocks preserve the spatial resolution and channel dimension of their corresponding stages.

For an input $x \in \mathbb{R}^{N\times3\times32\times32}$, the feature dimensions across the network are

$$[N, 3, 32, 32] \to [N, 16, 32, 32] \to [N, 32, 16, 16] \to [N, 64, 8, 8] \to [N, 64] \to [N, K], \quad \text{(A1)}$$

where N denotes the batch size and K denotes the number of classes. The Stem uses a 3×3 convolution with stride 1 to map the input to 16 feature channels. Layer1 preserves the 32×32 spatial resolution, while Layer2 and Layer3 reduce the spatial resolution to 16×16 and 8×8, respectively, and increase the number of channels to 32 and 64. At the end of the network, Global Average Pooling reduces each 8×8 feature map to a single channel feature, after which the classifier produces K class scores.

The stage-level network structure and ciphertext organization are summarized in Table A1.

Table A1: Network architecture and stage-level ciphertext layout of HELite-ResNet.

Stage	Structure	Feature size	Logical values	Ciphertext organization
Input	RGB image	$3 \times 32 \times 32$	3072	3 valid channel blocks; remaining slots are zero
Stem	3×3 Conv	$16 \times 32 \times 32$	16384	1×16384
Layer1	2 BasicBlocks	$16 \times 32 \times 32$	16384	1×16384
Layer2	HELite Transition + 2 BasicBlocks	$32 \times 16 \times 16$	8192	2×8192
Layer3	HELite Transition + 1 BasicBlock	$64 \times 8 \times 8$	4096	4×4096
Output	GAP + classifier	K	K	encrypted class scores

A.2 CKKS REPRESENTATION AND BASIC HOMOMORPHIC OPERATIONS

We employ CKKS SIMD packing for encrypted neural network inference. Each ciphertext contains 16,384 slots available for parallel computation. The input image is arranged in contiguous channel blocks, with each 32×32 input channel occupying 1,024 slots. The three RGB channels are stored in slots $0:1023$, $1024:2047$, and $2048:3071$, respectively, while the remaining slots are initialized to zero. After slot arrangement, the input vector is CKKS-encoded and encrypted to form the ciphertext input for server-side encrypted forward computation.

Server-side evaluation primarily uses ciphertext addition, ciphertext–plaintext multiplication, ciphertext–ciphertext multiplication, and ciphertext rotation. Ciphertext addition performs element-wise addition between corresponding slots and is used for intermediate accumulation and residual connections. Ciphertext–plaintext multiplication is used to multiply encrypted features by plaintext constants such as convolution weights, fixed masks, and scaling factors. In the execution graph considered in this work, ciphertext–ciphertext multiplication is primarily used for polynomial activation evaluation. Ciphertext rotation cyclically shifts SIMD slots to align spatial positions or channel contributions.

Before deployment, Batch Normalization parameters are fused offline into the weights and biases of the adjacent convolution. The fused parameters are then encoded as CKKS plaintexts according to the

corresponding slot organization, so BN is not evaluated separately in the online ciphertext execution graph. CKKS cannot directly evaluate the comparison operation required by ReLU; therefore, we approximate ReLU during encrypted inference using a degree-59 Chebyshev polynomial and evaluate it slot-wise using homomorphic additions and multiplications. Polynomial activation does not alter the channel ordering or slot positions of the current feature representation. Successive homomorphic multiplications gradually consume the available computation depth of a ciphertext. Bootstrapping is therefore applied at predetermined locations to restore the available levels required for subsequent encrypted computation while preserving the current logical tensor structure and slot organization. Because CKKS performs approximate numerical computation, encrypted inference may exhibit small numerical deviations from the corresponding plaintext computation.

A.3 Ciphertext Slot Layouts across Network Stages

HELite-ResNet uses CKKS ciphertexts with 16,384 slots throughout all network stages. Each stage adopts a contiguous channel layout determined by its channel dimension and spatial resolution, while the remaining slots are used to form repeated logical copies.

A.3.1 Stem and Layer1

The Stem output and all intermediate features in Layer1 have size $16 \times 32 \times 32 = 16384$. A complete logical feature therefore exactly fills one ciphertext. Each channel occupies $32 \times 32 = 1024$ contiguous slots, with spatial elements stored in row-major order. For spatial position (r, k) in channel c, the corresponding slot index is

$$\mathrm{slot}_{L1}(c, r, k) = 1024c + 32r + k, \tag{A2}$$

where $c \in \{0, \ldots, 15\}$ and $r, k \in \{0, \ldots, 31\}$.

Accordingly, channel 0 occupies slots $0 : 1023$, channel 1 occupies $1024 : 2047$, and the layout continues sequentially through channel 15 at $15360 : 16383$. Standard BasicBlocks in Layer1 do not change either the spatial resolution or the number of channels, and therefore preserve the same $16 \times 32 \times 32$ logical layout at both input and output.

A.3.2 Layer2

The logical output size of Layer2 is $32 \times 16 \times 16 = 8192$. Each channel contains $16 \times 16 = 256$ feature values, and one complete logical feature occupies 8,192 slots. Since a ciphertext contains 16,384 slots, Layer2 stores two identical logical copies within the same ciphertext.

For copy $q \in \{0, 1\}$, the slot corresponding to spatial position (r, k) in channel c is

$$\mathrm{slot}_{L2}(q, c, r, k) = 8192q + 256c + 16r + k, \tag{A3}$$

where $c \in \{0, \ldots, 31\}$ and $r, k \in \{0, \ldots, 15\}$.

Within each 8,192-slot group, channel 0 occupies relative slots $0 : 255$, channel 1 occupies $256 : 511$, and the layout continues sequentially through channel 31 at $7936 : 8191$. The first logical copy occupies slots $0 : 8191$, while the second occupies $8192 : 16383$. The subsequent two standard BasicBlocks in Layer2 preserve the $32 \times 16 \times 16$ feature size and therefore retain the same duplicated layout.

A.3.3 Layer3

The logical output size of Layer3 is $64 \times 8 \times 8 = 4096$. Each channel contains $8 \times 8 = 64$ feature values, and one complete logical feature occupies 4,096 slots. Consequently, a 16,384-slot ciphertext can store four identical copies of the Layer3 feature representation.

For copy $q \in \{0, 1, 2, 3\}$, the slot corresponding to spatial position (r, k) in channel c is

$$\mathrm{slot}_{L3}(q, c, r, k) = 4096q + 64c + 8r + k, \tag{A4}$$

where $c \in \{0, \ldots, 63\}$ and $r, k \in \{0, \ldots, 7\}$.

Within each 4,096-slot group, channel 0 occupies relative slots $0 : 63$, channel 1 occupies $64 : 127$, and the layout continues sequentially through channel 63 at $4032 : 4095$. The four logical copies

occupy slots $0:4095$, $4096:8191$, $8192:12287$, and $12288:16383$, respectively. The final standard BasicBlock in Layer3 does not change the spatial resolution or channel dimension and therefore preserves this four-copy layout.

At the stage level, the ciphertext organization of HELite-ResNet can be summarized as $1 \times 16384 \rightarrow 2 \times 8192 \rightarrow 4 \times 4096$. This organization maintains a regular channel arrangement within a fixed-capacity ciphertext across different spatial resolutions and provides a consistent data representation for subsequent ciphertext rotations, channel accumulation, and classification.

A.4 Residual Connections and Encrypted Classification Output

For a standard BasicBlock that preserves the feature dimensions, the Shortcut directly retains the input ciphertext. After the two convolutional operations in the main branch, its channel ordering and slot layout remain aligned with those of the Shortcut. Following the required numerical scale alignment, the two branches can therefore be combined through slot-wise ciphertext addition. The polynomial activation applied after residual addition preserves the logical slot structure of the current stage.

For the HELite Transition in Layer2 and Layer3, both the main branch and the Shortcut perform spatial downsampling and channel expansion. The main branch first reduces the spatial resolution and then applies convolution to produce the target number of channels. The Shortcut uses DownsampleA, which reduces the spatial resolution by sampling even rows and columns and pads the channel dimension with zeros. The two branches ultimately produce identical feature dimensions, channel ordering, and ciphertext slot layouts, enabling slot-wise residual addition.

The final Layer3 output contains 64 feature maps of size 8×8. Global Average Pooling first aggregates the 64 spatial positions within each channel using rotations and additions with steps $1, 2, 4, 8, 16$, and 32. The resulting ciphertext is then multiplied by a plaintext mask whose first slot in each 64-slot channel block is $1/64$ and whose remaining slots are zero, thereby retaining the corresponding channel-average feature. During classification, these encrypted features are multiplied by the encoded classification weights through ciphertext–plaintext multiplication, and cross-channel rotations and accumulation are used to aggregate the classification contributions from all channels, producing K encrypted class scores. The server returns the encrypted class scores to the client, which performs CKKS decryption and result recovery. Throughout server-side forward computation, the input image and all intermediate features remain encrypted.

Appendix B. Ciphertext Implementation of Homomorphic Convolution and Polynomial Activation

B.1 Execution of Homomorphic Convolution

Based on the ciphertext slot layouts described in Appendix A, the 3×3 convolutions in HELite-ResNet are mapped to ciphertext rotations, ciphertext–plaintext multiplications, and ciphertext additions supported by CKKS. For input channel i, output channel o, and output spatial position (r, k), the convolution after Batch Normalization (BN) fusion can be written as

$$Y_o(r,k) = \widetilde{b}_o + \sum_i \sum_{u,v\in\{-1,0,1\}} \widetilde{W}_{o,i,u,v} X_i(r+u, k+v). \tag{B1}$$

Ciphertext execution exploits the SIMD capability of CKKS to perform homomorphic convolution through a sequence of spatial alignment, plaintext weight multiplication, spatial accumulation, channel aggregation, and bias addition. The detailed spatial rotation, cyclic channel-diagonal weight encoding, and spatial and channel accumulation procedures are described in Appendices B.3–B.5.Figure B1 illustrates the complete ciphertext execution of a standard 3×3 convolution in Layer1.

B.2 Convolution–Batch Normalization Fusion and Plaintext Weight Encoding

To avoid evaluating Batch Normalization separately in the ciphertext domain, its parameters can be fused offline into the adjacent convolution before deployment. For output channel o, define

$$a_o = \frac{\gamma_o}{\sqrt{\sigma_o^2 + \epsilon}}, \tag{B2}$$

where γ_o, σ_o^2, and ϵ denote the Batch Normalization scaling parameter, running variance, and numerical stability term, respectively. When the convolution itself does not contain a bias term, the fused convolution weights and bias are

$$\widetilde{W}_{o,i,u,v} = a_o W_{o,i,u,v}, \qquad \widetilde{b}_o = \beta_o - a_o \mu_o, \tag{B3}$$

where μ_o and β_o denote the running mean and shift parameter, respectively. The original convolution and Batch Normalization can therefore be represented during encrypted inference by a fused convolution followed by a bias addition.

The fused convolution parameters are encoded as CKKS plaintexts according to the slot organization of the corresponding network stage. For each kernel position, weights are placed into the corresponding channel blocks according to the mapping between input and output channels, allowing one ciphertext–plaintext multiplication to operate on multiple spatial positions and channels in parallel. The fused bias is encoded according to the output-channel layout and repeated across the spatial slots of the corresponding channel. It can therefore be added directly after channel accumulation without modifying the ciphertext slot layout.

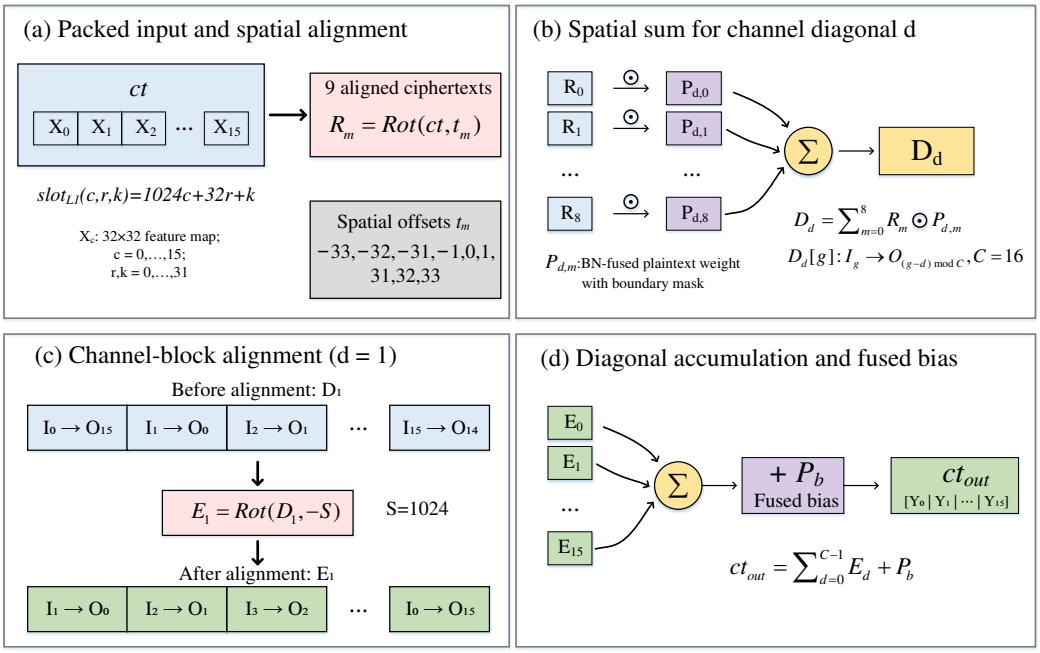


Figure B1: Ciphertext execution of a Layer1 standard 3×3 convolution with 16 input and 16 output channels: (a) spatial alignment, (b) per-diagonal spatial accumulation, (c) channel-block alignment, and (d) diagonal accumulation with fused bias.

B.3 Spatial Rotations for 3×3 Convolution

Under the contiguous channel layouts defined in Appendix A, spatial features within each channel are stored in row-major order. The nine neighboring positions required by a 3×3 convolution can therefore be obtained through fixed ciphertext rotations. For a feature map with spatial width W, the corresponding rotation offsets are $\{-W-1, -W, -W+1, -1, 0, 1, W-1, W, W+1\}$.

The rotation offsets used at different stages are summarized in Table B1.

Table B1: Spatial rotation offsets for 3×3 convolution at different network stages.

Stage	Spatial width	Rotation offsets
Layer1	32	$-33, -32, -31, -1, 0, 1, 31, 32, 33$
Layer2	16	$-17, -16, -15, -1, 0, 1, 15, 16, 17$
Layer3	8	$-9, -8, -7, -1, 0, 1, 7, 8, 9$

Let the input ciphertext be ct, and let the nine rotation offsets be $t_0, \ldots, t_8$. The spatially aligned ciphertexts are

$$R_m = \mathrm{Rot}(ct, t_m), \qquad m = 0, \ldots, 8. \tag{B4}$$

The zero offset corresponds to the center of the convolution kernel, while the remaining eight rotations align the upper-left, upper, upper-right, left, right, lower-left, lower, and lower-right neighbors, respectively. After spatial alignment, the nine ciphertexts are multiplied by the plaintext weights associated with the corresponding kernel positions, providing the inputs for subsequent spatial accumulation and channel aggregation. Boundary positions are handled by zeros pre-encoded into the plaintext weights, thereby preserving the convolution semantics of `padding=1`.

B.4 Cyclic Channel-Diagonal Encoding

Spatial rotations align neighboring positions within each channel, while cross-channel convolution requires the plaintext weights to be arranged according to the packed channel layout. For the d-th cyclic channel diagonal and the m-th kernel position, $P_{d,m}$ denotes a plaintext encoding of the BN-fused convolution weights with the corresponding boundary mask.

Assume that each channel occupies S contiguous CKKS slots, and let $g \in \{0, \ldots, C-1\}$ denote the physical channel-block index. Let I_g and O_o denote the input and output channel blocks, respectively. Within $P_{d,m}$, the weights stored in the g-th physical channel block correspond to the connection

$$I_g \to O_{(g-d) \bmod C}, \qquad g = 0, \ldots, C-1. \tag{B5}$$

Thus, d determines the cyclic input–output channel offset, while $m = 0, \ldots, 8$ identifies the corresponding kernel position. Boundary entries are zero-masked to preserve the `padding=1` convolution semantics described in Appendix B.3.

For the standard convolutions in Layer1, Layer2, and Layer3, the channel-block lengths are 1024, 256, and 64, respectively. Although the block length varies across stages, the same cyclic weight-encoding rule is used throughout.

B.5 Spatial Accumulation and Channel Accumulation

Let the nine spatially aligned ciphertexts be $R_0, \ldots, R_8$. For the d-th channel diagonal, the contributions from the nine kernel positions are first computed through ciphertext–plaintext multiplication and spatial accumulation:

$$D_d = \sum_{m=0}^{8} R_m \odot P_{d,m}, \tag{B6}$$

where $P_{d,m}$ is the cyclic channel-diagonal plaintext weight encoding defined in Appendix B.4.

At this point, D_d contains the complete 3×3 spatial convolution contribution for the d-th channel diagonal. Since ciphertext–plaintext multiplication and spatial accumulation do not alter the physical channel-block locations, the g-th physical channel block of D_d remains at input block I_g while logically corresponding to output block $O_{(g-d) \bmod C}$:

$$D_d[g] : I_g \to O_{(g-d) \bmod C}.$$

Let each channel block contain S slots. Under the rotation convention used in the current implementation, these contributions are aligned with their target output-channel blocks by

$$E_d = \mathrm{Rot}(D_d, -dS). \tag{B7}$$

After channel alignment, all channel diagonals are accumulated and the fused bias is added:

$$ct_{\mathrm{out}} = \sum_{d=0}^{C-1} E_d + P_b, \tag{B8}$$

where P_b denotes the encoded fused bias. After these operations, each output channel contains the convolution contributions accumulated from all input channels.

For example, a $16 \to 16$ convolution in Layer1 contains 16 channel diagonals, each associated with nine kernel positions. The convolution-weight computation therefore contains $16 \times 9 = 144$ ciphertext–plaintext multiplications. The standard convolutions in Layer2 and Layer3 use the same computational mechanism, differing only in the number of channels and the channel-block length.

B.6 Polynomial Activation and Bootstrapping

CKKS cannot directly evaluate the comparison operation required by ReLU. Encrypted inference therefore uses a degree-59 Chebyshev polynomial to approximate ReLU:

$$\mathrm{ReLU}(x) \approx P_{59}(x). \tag{B9}$$

Polynomial activation is evaluated independently in each SIMD slot and does not exchange data across spatial positions or channels. The feature dimensions, channel ordering, and slot layout therefore remain unchanged before and after activation. Polynomial evaluation involves ciphertext–ciphertext multiplications and progressively consumes the available computation levels, and is therefore coordinated with the Bootstrapping schedule.

To ensure that subsequent convolutions and polynomial activations can continue to be evaluated, Bootstrapping is applied at predetermined locations to restore the available levels required for later computation. For a standard BasicBlock that preserves the feature dimensions, the ciphertext execution order is Conv1+BN1 $\to$ Bootstrapping $\to P_{59} \to$ Conv2+BN2 $\to$ Shortcut scale alignment $\to$ Residual Add $\to$ Bootstrapping $\to P_{59}$.

After the second convolution, the Shortcut and main branch are first brought to compatible numerical scales before slot-wise residual addition. Neither Bootstrapping nor polynomial activation changes the logical feature dimensions or slot organization of the current stage.

Appendix C. Ciphertext Execution and Efficiency Analysis of HELite Transition

C.1 Execution of HELite Transition

HELite Transition restructures the ciphertext execution order of cross-resolution residual blocks. The original ResNet20 uses a 3×3 convolution with stride $= 2$ in the main branch to perform spatial downsampling and channel expansion simultaneously, while the Shortcut uses a 1×1 projection convolution with stride $= 2$. HELite Transition decouples these two operations: the main branch first performs 2×2 average pooling with stride $= 2$ followed by slot compaction, and then applies a 3×3 convolution with stride $= 1$ over the compact low-resolution layout. The Shortcut uses DownsampleA, which aligns with the main branch through even-position sampling, slot compaction, and zero-channel padding.

The key idea is to complete spatial downsampling and ciphertext compaction before channel expansion. This allows subsequent homomorphic convolution and channel alignment to operate directly on a smaller spatial layout while eliminating the projection convolution from the Shortcut.

C.2 Compact Ciphertext Layout within the Transition

C.2.1 Layer2 Transition

The Layer2 main branch receives encrypted features of shape $16 \times 32 \times 32$, with each input channel occupying 1024 consecutive slots. Average pooling uses the original ciphertext together with rotations by 1, 32, and 33 slots to align the four elements within each 2×2 window. Their sum is then multiplied by a plaintext mask with a fixed coefficient of 0.25. At this point, the pooled feature has a logical spatial size of 16×16, but its valid values remain sparsely distributed within the original channel blocks.

Column, row, and channel compaction are subsequently applied to rearrange the valid values into a contiguous layout. After compaction, each channel occupies 256 slots, and spatial position (r, k) in channel c is mapped to

$$\mathrm{slot}_{\mathrm{cmp}}(c, r, k) = 256c + 16r + k. \tag{C1}$$

Consequently, all $16 \times 16 \times 16 = 4096$ features are packed into one contiguous region, which is replicated four times across the 16384 slots to form the input layout for the subsequent $16 \to 32$ Conv1.

Conv1 is evaluated directly over this compact 16×16 layout. Its 32 output channels are divided into two ciphertext halves, responsible for output channels 0–15 and 16–31, respectively. Both halves use all 16 input channels, and the convolution is evaluated through spatial rotations, plaintext-weight multiplications, and channel-block accumulation.

The Layer2 Shortcut uses DownsampleA, whose logical mapping is

$$Y_o^{\mathrm{sc}}(r, k) = \begin{cases} X_o(2r, 2k), & 0 \le o < C_{\mathrm{in}}, \\ 0, & C_{\mathrm{in}} \le o < C_{\mathrm{out}}. \end{cases} \tag{C2}$$

Here, $C_{\mathrm{in}} = 16$ and $C_{\mathrm{out}} = 32$. On ciphertexts, a fixed $0/1$ plaintext mask retains the even-indexed rows and columns, after which the valid values are compacted. In each 8192-slot period, the first 4096 slots contain the 16 valid channels, while the remaining 4096 slots correspond to the 16 newly introduced zero channels. This layout is replicated once to match the main-branch output. DownsampleA contains no trainable convolution weights or BN parameters.

C.2.2 Layer3 Transition

Layer3 follows the same execution principle. Each 16×16 input channel occupies 256 slots, and average pooling uses rotations by 1, 16, and 17 slots. After spatial and channel compaction, each 8×8 channel occupies only 64 slots, allowing the $32 \times 8 \times 8 = 2048$ valid features to be stored contiguously. Before Conv1, this 2048-slot compact block is replicated eight times to form the input layout for the subsequent $32 \to 64$ homomorphic convolution.

The 64 output channels of Conv1 are divided into two ciphertext halves, each containing 32 output channels, or 2048 valid slots. After the two halves are merged, they form a complete $64 \times 8 \times 8 = 4096$ feature group, which is replicated four times across the 16384-slot ciphertext.

Layer3 DownsampleA likewise retains the even-indexed rows and columns of the input. After compaction, the first 2048 slots contain 32 valid 8×8 channels, while the following 2048 slots correspond to the 32 newly introduced zero channels, producing a 4096-slot group aligned with the main branch.

C.3 Level Consumption and Bootstrapping Scheduling

In CKKS, the available computational depth of a ciphertext is constrained by the modulus chain. Consecutive homomorphic multiplications along the critical path, together with the associated rescaling operations, progressively consume the available levels, whereas ciphertext rotations, ciphertext additions, and slot replication implemented solely by rotations and additions do not increase the

Table C1: Intermediate compact layouts of the Layer2 and Layer3 HELite Transitions.

Item	Layer2	Layer3
Slots per input channel	1024	256
Slots per channel after downsampling	256	64
Compact feature before Conv1	$16 \times 16 \times 16$	$32 \times 8 \times 8$
Compact block size	4096	2048
Copies before Conv1	4	8
Conv1	$16 \to 32$	$32 \to 64$
Layout after Conv1	2×8192	4×4096

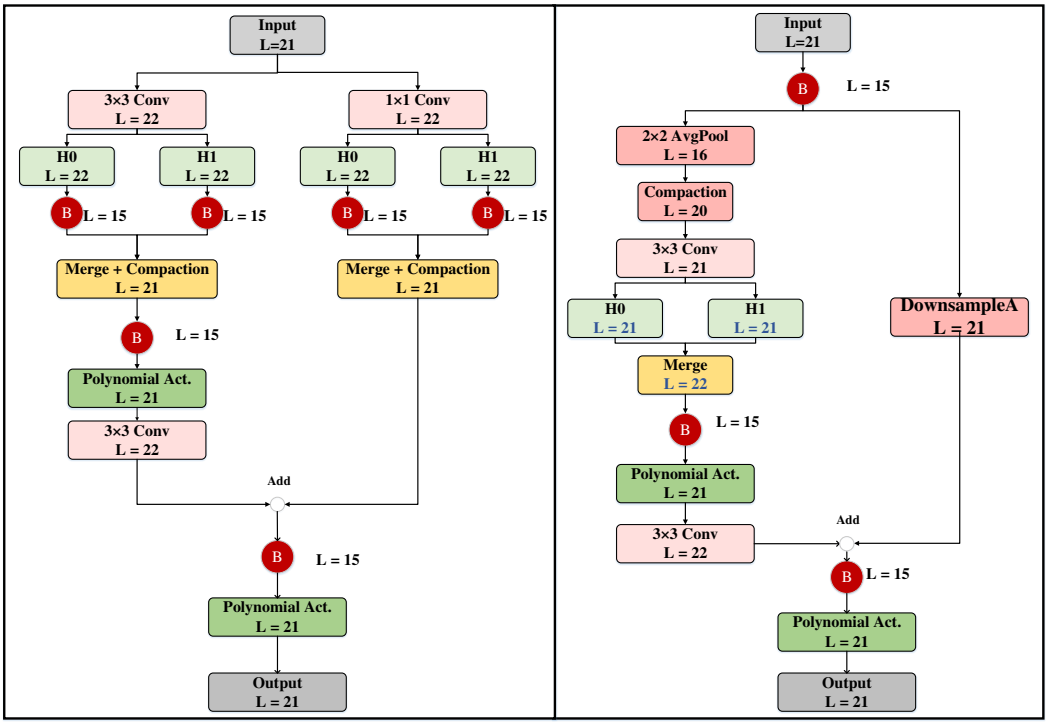


Figure C1: Ciphertext-level execution, level consumption, and bootstrapping schedules of (a) the original ResNet20 transition and (b) HELite Transition. HELite Transition reduces the number of bootstrapping operations from six to three.

multiplicative depth. Multiple independent plaintext-mask multiplications applied at the same input level can be evaluated in parallel; therefore, their contribution to the critical path is determined by the number of serial multiplication stages rather than by the total number of plaintext-mask multiplications.

In the following execution graph, L denotes the level index used by our implementation rather than the number of remaining levels. A larger value of L indicates that more multiplicative levels have been consumed. The red node B denotes Bootstrapping, which refreshes the ciphertext and resets its level index to $L = 15$. H0 and H1 denote the two ciphertext halves generated by a channel-expanding convolution.

As illustrated in Figure C1, the original Transition independently bootstraps the ciphertext halves from the main branch and projection Shortcut before layout compaction and residual fusion, resulting in six Bootstraps. HELite moves spatial downsampling and slot compaction before channel expansion, shares the input Bootstrap between the main branch and DownsampleA, and reduces the number of Bootstraps from six to three.

For the Layer2 Transition of the original ResNet20, the stride-2 Conv1 transforms an input of $16 \times 32 \times 32$ into an output of $32 \times 16 \times 16$. Because the convolution results remain organized according to the original 32×32 slot coordinates, the 32 output channels are carried by two ciphertext halves: the first half stores output channels 0–15, whereas the second half stores output channels 16–31. After the two halves are bootstrapped independently, `downsample(0)` performs half placement, spatial compaction, and channel compaction. The resulting critical-path depth is

$$\begin{aligned} D_{\text{orig-down}} &= D_{\text{half}} + D_{\text{col}}^{\text{orig}} + D_{\text{row}} + D_{\text{ch}} \\ &= 1 + 3 + 1 + 1 = 6. \end{aligned} \tag{C3}$$

Here, $D_{\text{half}} = 1$ corresponds to the slot placement of the two ciphertext halves. Each half is multiplied by a complementary plaintext mask so that output channels 0–15 and 16–31 are placed into disjoint target regions, after which the two masked ciphertexts are combined by ciphertext addition. Since the two Ct-Pt multiplications are independent, they can be evaluated at the same multiplicative level and therefore contribute only one level to the critical path; the subsequent ciphertext addition does not increase the multiplicative depth.

The dominant serial dependency in the original `downsample(0)` arises from column-wise compaction. For an arbitrary valid output row, the initial sparse layout can be written as $[a_0, 0, a_1, 0, \ldots, a_{15}, 0]$, while the target layout is $[a_0, a_1, \ldots, a_{15}, 0, \ldots, 0]$. The original implementation performs progressive doubling:

$$\begin{aligned} ct^{(1)} &= (ct + \mathrm{Rot}(ct, 1)) \odot M_2, \\ ct^{(2)} &= \Big(ct^{(1)} + \mathrm{Rot}(ct^{(1)}, 2)\Big) \odot M_4, \\ ct^{(3)} &= \Big(ct^{(2)} + \mathrm{Rot}(ct^{(2)}, 4)\Big) \odot M_8, \\ ct_{\text{col}} &= ct^{(3)} + \mathrm{Rot}(ct^{(3)}, 8). \end{aligned} \tag{C4}$$

The masks M_2, M_4, and M_8 successively retain contiguous valid regions of lengths 2, 4, and 8, respectively. The corresponding data organization can be summarized as $1 \to 2 \to 4 \to 8 \to 16$. Because each mask multiplication depends on the output of the preceding stage, the first three stages form a strict serial dependency, yielding $D_{\text{col}}^{\text{orig}} = 3$. The final $8 \to 16$ concatenation uses only rotation and addition and therefore does not consume an additional level. Row compaction subsequently contributes one level through parallel row-mask selection and rotation, while channel compaction contributes one additional level through parallel channel-mask operations.

HELite places AvgPool and Slot Compaction before the channel-expanding Conv1. The average-pooling operation first aggregates the four slots within each 2×2 window using rotations and additions, and then multiplies the result by a plaintext mask that encodes both the averaging coefficient and the sampling positions. Consequently, the pooling stage has depth $D_{\text{pool}} = 1$. After AvgPool, each output row remains sparsely arranged. HELite first performs one adjacent-element pairing step and then directly gathers all column pairs in parallel:

$$\begin{aligned} ct^{(0)} &= (ct_{\text{pool}} + \mathrm{Rot}(ct_{\text{pool}}, 1)) \odot M_2, \\ ct_{\text{col}} &= \sum_{p=0}^{7} \mathrm{Rot}\Big(ct^{(0)}, 2p\Big) \odot M_p^{\text{col}}. \end{aligned} \tag{C5}$$

The first operation transforms $[a_0, 0, a_1, 0, a_2, 0, a_3, 0, \ldots]$ into $[a_0, a_1, 0, 0, a_2, a_3, 0, 0, \ldots]$, consuming one level. The masks M_p^{col} then select the corresponding column pairs and place (a_0, a_1), (a_2, a_3), …, (a_{14}, a_{15}) directly into their target positions. Because all column-pair operations are independently derived from the same $ct^{(0)}$, the associated Ct-Pt multiplications can be evaluated at the same multiplicative level. The column-compaction depth is therefore reduced to $D_{\text{col}}^{\text{HELite}} = 2$, one level lower than in the original implementation.

After column compaction, row compaction and channel compaction each contribute one additional level. The complete HELite Slot Compaction therefore has depth $D_{\text{comp}} = 2 + 1 + 1 = 4$. AvgPool

introduces one new level, while the serial depth of column compaction is reduced from 3 to 2. Accordingly, AvgPool together with column compaction still requires $1 + 2 = 3$ levels, matching the three levels required by the original column-compaction stage.

After Slot Compaction, the main branch executes a $16 \to 32$ Conv1 on the compact 16×16 layout. The Ct-Pt weight multiplication in Conv1 contributes one level to the critical path and produces two ciphertext halves carrying output channels 0–15 and 16–31, respectively. The two output halves are then multiplied by plaintext masks that place them into disjoint slot regions and are subsequently combined by ciphertext addition. This stage therefore contributes one additional level to the critical path.

Consequently, from the Bootstrap at the entrance of the HELite Transition to the next Bootstrap after merging the Conv1 output halves, the critical-path depth of the main branch is

$$\begin{aligned} D_{\text{HELite-window}} &= D_{\text{pool}} + D_{\text{comp}} + D_{\text{conv1}} + D_{\text{merge}} \\ &= 1 + 4 + 1 + 1 = 7. \end{aligned} \tag{C6}$$

Under the level convention used in the current implementation, this computation window can be expressed as

$$15 \xrightarrow[+1]{\text{AvgPool}} 16 \xrightarrow[+4]{\text{Slot Compaction}} 20 \xrightarrow[+1]{\text{Conv1}} 21 \xrightarrow[+1]{\text{Half Merge}} 22 \xrightarrow{\text{Bootstrap}} 15. \tag{C7}$$

This level sequence corresponds to the main branch in Figure C1: the input Bootstrap resets the ciphertext to $L = 15$, AvgPool, Slot Compaction, Conv1, and Half Merge advance the level index to $L = 22$, and the following Bootstrap resets it to $L = 15$.

This execution order allows the ciphertext refreshed at the Transition entrance to complete AvgPool, Slot Compaction, low-resolution Conv1, and output-half merging before the next Bootstrap is applied at the end of the available computation window. In the original implementation, Conv1 first produces two high-resolution output halves that must be bootstrapped independently before subsequent downsampling and merging. In HELite, a single Bootstrap is performed at the Transition entrance and shared by the main branch and Shortcut; after compact Conv1, the two output halves are merged first and then refreshed together by a single Bootstrap.

The Shortcut branch uses the parameter-free DownsampleA. It directly selects the even rows and even columns from the shared refreshed input and compacts the retained values into the same low-resolution layout as the main branch. Its critical-path depth is

$$\begin{aligned} D_{\text{DownsampleA}} &= D_{\text{select}} + D_{\text{col}}^{\text{orig}} + D_{\text{row}} + D_{\text{ch}} \\ &= 1 + 3 + 1 + 1 = 6. \end{aligned} \tag{C8}$$

The first level is consumed by a fixed $0/1$ plaintext mask that selects the even rows and columns. This is followed by three serial column-compaction stages implemented with `gen_mask(2)`, `gen_mask(4)`, and `gen_mask(8)`, contributing three levels in total. Row compaction and channel compaction each contribute one further level. The final layout replication consists only of rotations and additions and therefore does not increase the multiplicative depth. Although DownsampleA retains a 6-level slot-rearrangement path, it eliminates the 1×1 projection convolution used in the original Shortcut, including its convolution-weight multiplications, channel-wise weighted accumulation, and the independent Bootstraps required by its two projection output halves.

Considering both the main branch and the Shortcut, the original Transition performs six Bootstraps: two for the main-branch Conv1 output halves, one after the main-branch `downsample(0)`, two for the projection-Shortcut output halves, and one after residual addition. HELite performs three Bootstraps: one shared Bootstrap at the Block entrance, one after merging the compact Conv1 output halves, and one after residual addition. Thus, while preserving the multiplicative depth required by the cross-resolution computation, HELite reorganizes the placement of downsampling, slot compaction, and output-half merging to reduce independent ciphertext refreshes and lower the overall homomorphic execution cost of the Transition.

Table C2: Local homomorphic operation comparison between the original Transition and HELite Transition.

Metric	Layer2 Original	Layer2 HELite	Layer3 Original	Layer3 HELite
Rotations	235	171	389	254
Ct-Pt Mult.	714	657	1368	1241
Bootstraps	6	3	6	3

C.4 FHE Cost Reduction of HELite Transition

The efficiency gains of HELite Transition mainly arise from its compact-before-expansion execution order and the replacement of the projection Shortcut with DownsampleA. Table C2 summarizes the major homomorphic operation changes in the two cross-resolution blocks.

The reduction in rotations mainly comes from the Shortcut and the earlier spatial compaction. The number of rotations decreases from 235 to 171 in Layer2 and from 389 to 254 in Layer3, while the corresponding ciphertext–plaintext multiplications decrease from 714 to 657 and from 1368 to 1241, respectively. In addition, the number of explicit Bootstrapping operations per Transition is reduced from 6 to 3. HELite performs one shared refresh at the Transition input, followed by one refresh after merging the Conv1 halves and another after residual addition, thereby avoiding the independent Bootstrapping operations required by the two output halves of the projection Shortcut.

By restructuring the cross-resolution execution path, HELite Transition substantially reduces homomorphic operations, ciphertext refreshes, and layout-conversion overhead, thereby improving the overall efficiency of encrypted inference.

Appendix D. Architecture Compression and Knowledge Distillation

D.1 Network Compression and Homomorphic Computation Cost

The main text adopts $[2, 3, 2]$ as the final stage configuration of HELite-ResNet. To further analyze the relationship between network depth and FHE execution cost, we additionally consider the $[2, 2, 2]$ configuration. Both HELite configurations use the same HELite Transition, while $[2, 2, 2]$ is obtained by further removing one standard BasicBlock from Layer2 of $[2, 3, 2]$. The homomorphic operation counts follow the same convention as in the main text and use the degree-59 Chebyshev polynomial activation setting.

Table D1: Comparison of homomorphic operations across three stage-depth configurations.

Structure	Act.	Ct-Pt	Ct-Ct	Rot.	Boot.
ResNet20 $[3, 3, 3]$	19	6,571	304	1,344	27
HELite $[2, 3, 2]$	15	4,945	240	937	17
HELite $[2, 2, 2]$	13	4,368	208	851	15
$[2, 2, 2]$ vs. $[2, 3, 2]$	-13.3%	-11.7%	-13.3%	-9.2%	-11.8%

Note. Act. denotes the number of polynomial activations; Ct-Pt and Ct-Ct denote the numbers of ciphertext–plaintext and ciphertext–ciphertext multiplications, respectively; Rot. denotes the number of ciphertext rotations; and Boot. denotes the number of Bootstrapping operations.

Compared with the original $[3, 3, 3]$ configuration, $[2, 3, 2]$ already substantially reduces the major homomorphic operations. Further compressing the network to $[2, 2, 2]$ yields an additional reduction of approximately 9%–13% across the main operation categories. Therefore, removing additional standard BasicBlocks can further reduce encrypted inference cost, although the incremental gain is relatively limited.

D.2 Performance Trade-off between Network Depth and Knowledge Distillation

Network compression directly reduces homomorphic convolutions, polynomial activations, and Bootstrapping operations, but excessive reduction in network depth may affect the feature representation capability of the model. To analyze the effect of knowledge distillation under different compression levels, we compare the plaintext classification performance of $[2, 3, 2]$ and $[2, 2, 2]$ with and without knowledge distillation.

During training, a frozen teacher network provides soft targets, while the student model is optimized using both ground-truth supervision and the distillation objective. The teacher network is used only during training, and only the student model is retained for encrypted deployment. Therefore, for the same student architecture, the KD and no-KD configurations have identical online homomorphic operation counts.

For the $[2, 3, 2]$ configuration, knowledge distillation improves RAF-DB accuracy from 83.05% to 84.45%, increases FERPlus best-val from 86.28% to 86.62%, and improves CIFAR-10 accuracy from 91.64% to 92.95%. FERPlus test@best-val changes from 84.87% to 84.84% and therefore remains essentially unchanged. Since knowledge distillation is used only during training, this configuration does not increase the homomorphic operation cost of the student model during encrypted inference.

For the $[2, 2, 2]$ configuration, knowledge distillation improves FERPlus best-val, RAF-DB, and CIFAR-10, while FERPlus test@best-val decreases from 84.87% to 84.30%. Compared with the knowledge-distilled $[2, 3, 2]$ configuration, $[2, 2, 2]$ is approximately 0.91, 0.54, and 0.67 percentage points lower on FERPlus best-val, FERPlus test@best-val, and CIFAR-10, respectively, while the difference on RAF-DB is only about 0.03 percentage points. Therefore, although further compression to $[2, 2, 2]$ reduces FHE cost, its cross-dataset performance preservation is less consistent than that of $[2, 3, 2]$.

Table D2: Plaintext classification performance under different network depths and knowledge distillation settings.

Model	KD	CIFAR-10	FERPlus Val	FERPlus Test	RAF-DB
ResNet20 $[3, 3, 3]$	No	92.80	86.28	84.94	83.90
HELite $[2, 3, 2]$	No	91.64	86.28	84.87	83.05
HELite $[2, 3, 2]$	Yes	92.95	86.62	84.84	84.45
HELite $[2, 2, 2]$	No	91.83	85.65	84.87	83.02
HELite $[2, 2, 2]$	Yes	92.28	85.71	84.30	84.42

Note. FERPlus Val denotes the best validation accuracy, while FERPlus Test denotes the test accuracy evaluated at the checkpoint with the best validation accuracy. All results are reported in percent (%). The $[2, 3, 2]$ configurations without and with KD correspond to the Without KD and HELite-ResNet variants, respectively, in Table 3.

D.3 HELite-ResNet Architecture Selection

Considering both homomorphic execution cost and classification performance, $[2, 3, 2]$ is selected as the default HELite-ResNet configuration. This structure already achieves the major FHE cost reduction relative to the original $[3, 3, 3]$ network while maintaining relatively stable classification performance across multiple datasets.

Further compression to $[2, 2, 2]$ can reduce the major homomorphic operations by an additional approximately 9%–13%, but the incremental efficiency gain is relatively limited, while performance degradation becomes more noticeable on some datasets. Therefore, $[2, 3, 2]$ provides a more balanced trade-off between FHE inference efficiency and classification performance.

Knowledge distillation serves as a training-stage performance enhancement mechanism and does not alter the ciphertext execution graph of the student model. It can therefore improve the classification performance of the compressed model across multiple datasets while preserving the established FHE computation cost.