SCALABLE MULTI-USER SECURE TRANSFORMER INFERENCE WITH SSE-ENABLED THRESHOLD CKKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Homomorphic encryption (HE) enables privacy-preserving transformer inference on encrypted inputs, but existing systems largely assume a single cryptographic owner. Extending HE inference to independent users therefore requires reusable collective cryptographic infrastructure. For transformer inference, we further employ sparse-secret encapsulation (SSE) to make CKKS bootstrapping more level-efficient. However, SSE introduces a fixed-weight sparse secret in addition to the dense secret used for regular computation, making distributed setup substantially more challenging. We develop an SSE-enabled threshold-CKKS framework centered on a reusable two-server-assisted setup. It generates dense and sparse collective secrets while preserving their prescribed distributions and structures, realizes both first in the Boolean-sharing domain, and delays RNS conversion until distributed CKKS and SSE key generation. Built on this infrastructure, ciphertexts from multiple independent users are consolidated through interleaved packing for shared transformer evaluation, followed by per-user extraction and synchronized threshold decryption. With 16 users, shared inference reduces service time by $15.998\times$ with only 0.00064% interface overhead, while maintaining task accuracy within 0.3 percentage points of plaintext inference and producing identical predictions to single-user encrypted inference.

1 INTRODUCTION

Transformer-based large language models (LLMs) like ChatGPT Achiam et al. (2023) are increasingly deployed as cloud services, raising privacy concerns over sensitive user inputs. Secure transformer inference (STI) addresses these concerns using cryptographic techniques to protect user data.

Related work on STI. Existing STI can be divided into interactive approaches Pang et al. (2024); Lu et al. (2025); Li et al. (2024) that combine fully homomorphic encryption (FHE) with secure multiparty computation (MPC) and non-interactive approaches Zhang et al. (2025); Moon et al. (2025); Zhang et al. (2026) that perform inference entirely under FHE. Most existing systems, however, are designed and optimized for a single user and cryptographic context. Extending interactive approaches to concurrent users further amplifies communication and synchronization overhead, as users must repeatedly participate during inference. We therefore focus on non-interactive FHE inference and adopt CKKS for efficient SIMD computation over real-valued vectors.

Extending CKKS-based inference to multiple independent users requires more than processing additional ciphertexts in parallel. Independent cryptographic contexts place user ciphertexts in different key domains, preventing them from sharing the same packing and inference pipeline. Efficient multi-user inference therefore requires a *common cryptographic infrastructure* with compatible CKKS parameters, a collective public-key domain, and coherent evaluation keys.

Table 1: Bootstrapping comparison at $N = 2^{16}$ under 128-bit security.

Config.	QP (bits)	RMSE-eq. (bits)	Level	TPS_{GPU} (ct/s)
Dense	1743	≈ 11.77	11	1.453
SSE	1740	≈ 12.04	**21**	1.414

Threshold CKKS supports distributed key ownership, but secure transformer inference still requires repeated bootstrapping. For instance, in MOAI Zhang et al. (2026)'s LLaMA3-8B evaluation, the Feed-Forward Network (FFN) expands the encrypted state from 4,096 to 14,336 ciphertexts under column packing while consuming 13 levels. With only 11 post-bootstrap levels, dense-secret bootstrap-

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.

ping requires a refresh within this widened state, whereas sparse-secret encapsulation (SSE) Bossuat et al. (2022) provides 21 levels, as shown in Table 1, and avoids this costly wide-state bootstrap. Realizing this benefit in a multi-user threshold setting, however, requires distributed generation of heterogeneous collective secrets and cross-secret key material, raising three key challenges:

C1: Distribution-Preserving Secret Generation. SSE requires collective dense and sparse secrets with prescribed distributions. Conventional threshold-CKKS constructions Mouchet et al. (2021); Lee et al. (2023); Badawi et al. (2022); Hu et al. (2025) aggregate independently sampled local secrets, which does not preserve these distributions, particularly the fixed Hamming weight required by the sparse secret. The DKG Min et al. (2025) can generate the sparse secret, but relies on a pre-established threshold-CKKS key infrastructure. **C2: RNS-Consistent Cross-Domain Conversion.** Secret generation relies on Boolean-friendly operations such as rejection sampling, equality testing, and collision detection, whereas CKKS key generation requires RNS-consistent arithmetic shares. Premature RNS realization would replicate intermediate states across all limbs, increasing computation and communication. **C3: Scalable Setup with Distributed Ownership.** Involving all users in these secure computations introduces party-dependent interaction and synchronization, whereas delegating the setup to a single server exposes the collective secrets. The setup must therefore remain lightweight for users while preserving distributed ownership of the long-term decryption secret.

Our key insight is to *decouple distributed ownership of the common cryptographic infrastructure from its expensive construction*. Two non-colluding auxiliary servers directly generate the collective secrets in the Boolean domain, delay RNS conversion until their structures are fixed, and generate the SSE-enabled threshold CKKS key material without reconstructing either secret. The common SSE-enabled threshold CKKS domain enables joint packing and evaluation of independently encrypted user inputs. We extend interleaved batching Zhang et al. (2026) along the user dimension for multi-user transformer inference without cross-key conversion. Inference uses the collective dense secret, switching temporarily to the sparse secret only during bootstrapping, and outputs are recovered via synchronized threshold decryption Colin de Verdière et al. (2026). In summary, this work makes the following contributions. A detailed discussion of related work is provided in Appendix A.

- **Scalable multi-user secure inference.** We develop an SSE-enabled threshold-CKKS framework for multi-user secure transformer inference. A two-server-assisted setup establishes reusable collective cryptographic infrastructure without a trusted dealer.
- **Distribution-preserving and RNS-consistent setup.** We generate collective dense and fixed-weight sparse secrets in the Boolean domain while preserving their prescribed distributions and structures. We delay RNS realization and derive RNS-consistent threshold-CKKS and SSE key material without reconstructing either secret.
- **Multi-user interleaved inference.** We consolidate fragmented requests from independent users into one shared interleaved transformer evaluation while preserving per-user outputs. With 16 users sharing one packing capacity, this reduces service time by $15.998\times$ over separate execution with only 0.00064% interface overhead and no prediction changes from single-user inference.

2 PRELIMINARIES

2.1 MULTI-USER THRESHOLD CKKS AND SSE

In multi-user secure inference, ciphertexts from different parties must share a common CKKS domain. Threshold CKKS establishes this domain using a collective secret $s = \sum_{m=1}^{M} s^{(m)} \in R_Q$, whose shares remain distributed among the parties, while the corresponding public and evaluation keys are generated jointly without reconstructing s. Each party P_m can then encrypt its input under the collective public key pk as $ct_m = \mathsf{Enc}_{pk}(\mathsf{Encode}(x_m))$. However, aggregating independently sampled local secrets does not generally preserve a prescribed global secret distribution.

SSE further introduces two secret domains, $s_d \leftarrow \chi_{\text{dense}}$ and $s_s \leftarrow \chi_{\text{sparse}}$, where $\mathrm{HW}(s_s) = h$. s_d supports regular homomorphic evaluation and s_s is used temporarily around modulus raising:

$$ct^q_{s_d} \xrightarrow{K^{qp}_{d\to s}} ct^q_{s_s} \xrightarrow{\mathsf{ModRaise}} ct^Q_{s_s} \xrightarrow{K^{QP}_{s\to d}} ct^Q_{s_d}. \tag{1}$$

This separation allows the normal computation to retain the dense-secret modulus budget while using the sparse secret only where required by bootstrapping. An SSE-enabled threshold deployment

must therefore establish distribution-correct collective secrets s_d and s_s, together with coherent public, evaluation, and cross-secret switching keys.

2.2 MPC PRELIMINARIES

We briefly review the Boolean and arithmetic secret-sharing representations used throughout the paper. Let $\mathcal{C} = \{C_0, C_1\}$ denote two computing parties.

Boolean Sharing. For a bit $x \in \{0, 1\}$, its Boolean sharing is denoted by $[\![x]\!]^{\mathcal{C}}_{\mathsf{B}} = (x^{(0)}_{\mathsf{B}}, x^{(1)}_{\mathsf{B}})$, and $x = x^{(0)}_{\mathsf{B}} \oplus x^{(1)}_{\mathsf{B}}$. Boolean circuits operate on bit-level representations. XOR gates ($\oplus$) and NOT gates ($\neg$) are evaluated locally on Boolean shares, whereas non-linear gates such as AND ($\wedge$) require secure interaction. Boolean representations are therefore commonly used for bit-oriented operations such as equality tests, comparisons, validation, and selection. For instance, for two n-bit Boolean-shared values $[\![\mathbf{x}]\!]^{\mathcal{C}}_{\mathsf{B}}$ and $[\![\mathbf{y}]\!]^{\mathcal{C}}_{\mathsf{B}}$, equality $\mathsf{EQ}([\![\mathbf{x}]\!]^{\mathcal{C}}_{\mathsf{B}}, [\![\mathbf{x}]\!]^{\mathcal{C}}_{\mathsf{B}}) = \bigwedge_{i=0}^{n-1} \neg([\![x_i]\!]_{\mathsf{B}} \oplus [\![y_i]\!]_{\mathsf{B}})$ computes the bitwise XOR followed by NOT and aggregates the resulting bits with a balanced Boolean AND tree, requiring $n - 1$ AND gates and an interaction depth of $\lceil \log_2 n \rceil$.

Arithmetic Sharing. For $x \in \mathbb{Z}_q$, an arithmetic sharing is denoted by $[\![x]\!]^{\mathcal{C}}_q = (x^{(0)}_q, x^{(1)}_q)$, and $x = x^{(0)}_q + x^{(1)}_q \pmod q$. Arithmetic circuits operate directly over the underlying ring $\mathbb{Z}_q$. Addition of arithmetic shares is local, whereas multiplication generally requires interaction. This representation is suitable for modular and ring-arithmetic computations. For an RNS basis $\mathcal{B} = \{q_0, \ldots, q_{L_{\mathcal{B}}-1}\}$, we write $[\![x]\!]^{\mathcal{C}}_{\mathcal{B}} = \{[\![x]\!]^{\mathcal{C}}_{q_\ell}\}_{\ell=0}^{L_{\mathcal{B}}-1}$ for an RNS-consistent arithmetic sharing, where all limbs represent the same underlying value x.

3 SYSTEM AND DESIGN OVERVIEW

This section presents the system setting and an overview of our design. The threat model is summarized in Appendix B.

3.1 SYSTEM MODEL

As shown in Fig 1, we consider a multi-user secure transformer inference system with M parties $\mathcal{P} = \{P_m\}_{m=1}^M$, two auxiliary setup servers $\mathcal{C} = \{C_0, C_1\}$, and an inference server $\mathcal{S}$. The system consists of a distributed setup phase and an online inference phase.

During setup, each party P_m contributes private randomness ρ_m, while C_0 and C_1 jointly establish a SSE-enabled threshold CKKS infrastructure. The setup outputs public key material $\mathcal{K}_{\mathrm{pub}}$ and a long-term dense-secret share $s_d^{(m)}$ for each party, satisfying $s_d = \sum_{m=1}^M s_d^{(m)}$. The sparse secret s_s is used only for SSE bootstrapping and is not retained long term. The auxiliary servers retain no secret-key state after setup.

During online inference, each party submits ct_m to $\mathcal{S}$, and $\mathcal{S}$ evaluates them through a shared encrypted transformer inference pipeline using $\mathcal{K}_{\mathrm{pub}}$. The output for P_m is recovered using the adopted synchronized threshold-decryption protocol Colin de Verdière et al. (2026). The auxiliary servers participate in neither online evaluation nor final decryption.

3.2 DESIGN OVERVIEW

Under the model above, the two auxiliary servers execute a three-stage setup consisting of distribution-preserving secret generation, delayed RNS realization, and key-material generation with ownership handoff. These stages address C1–C3 and establish the common cryptographic domain used during online inference, as summarized in Figure 1.

The parties first contribute private randomness $\{\rho_m\}_{m=1}^M$. The auxiliary servers use it to construct the collective dense and sparse secrets under Boolean sharing and then realize only their final representations over the required RNS bases:

$$\{\rho_m\}_{m=1}^M \xrightarrow{\mathsf{Gen}_{\mathsf{B}}} \left\{([\![P_x]\!]^{\mathcal{C}}_{\mathsf{B}}, [\![M_x]\!]^{\mathcal{C}}_{\mathsf{B}})\right\}_{x\in\{d,s\}} \xrightarrow{\mathsf{B2RNS}} \left\{[\![s_x]\!]^{\mathcal{C}}_{\mathcal{B}}\right\}_{x\in\{d,s\}}. \tag{2}$$

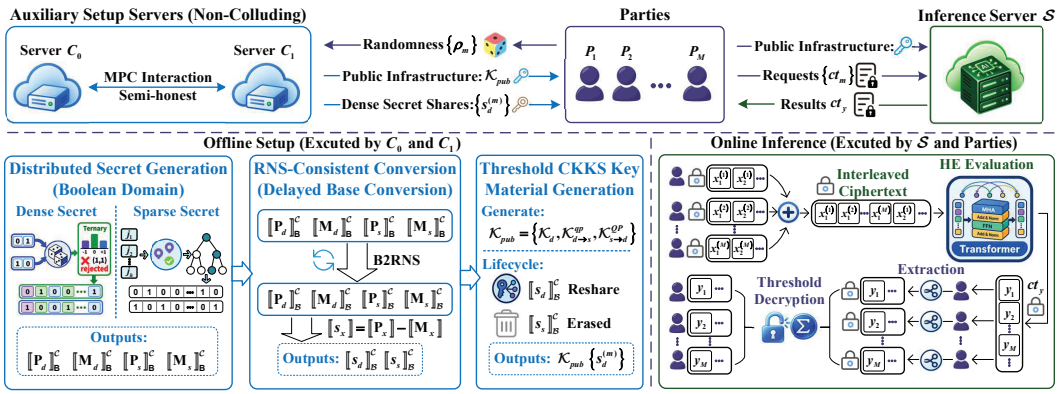

Figure 1: System architecture and phase-separated workflow.

Here, $\mathsf{Gen}_{\mathsf{B}}$ directly constructs a dense ternary secret and a balanced fixed-weight sparse secret according to their prescribed distributions. The subsequent RNS realization is delayed until their positive and negative indicator vectors are fixed. In particular, the h sparse one-hot vectors are first combined into P_s and M_s, reducing the Boolean state subjected to RNS realization from hN to $2N$. The resulting RNS shares are used along two paths:

$$\{[\![s_x]\!]^{\mathcal{C}}_{\mathcal{B}}\}_{x\in\{d,s\}} \xrightarrow{\mathsf{KeyGen}} \mathcal{K}_{\mathrm{pub}}, \quad [\![s_d]\!]^{\mathcal{C}}_{\mathcal{B}_Q} \xrightarrow{\mathsf{ReShare}_{2\to M}} \{s_d^{(m)}\}_{m=1}^{M}. \tag{3}$$

The first path produces key materials $\mathcal{K}_{\mathrm{pub}} = (\mathcal{K}_d, \mathcal{K}^{qp}_{d\to s}, \mathcal{K}^{QP}_{s\to d})$, where $\mathcal{K}_d$ contains the standard CKKS keys and the other two components provide the SSE cross-secret switching keys. The second path transfers the same dense secret into the M-party decryption domain. The parties additionally establish the private state required by synchronized threshold decryption. The auxiliary servers then erase their temporary secret shares and leave the protocol.

During online inference, active parties independently encrypt their inputs under the collective public key. Their ciphertexts can be consolidated directly in the adopted interleaved layout without cross-key conversion. The inference server evaluates the transformer over the packed ciphertext, after which the protocol-determined per-user outputs are recovered through synchronized threshold decryption. The underlying transformer operators remain unchanged, and the proposed setup enables them to process ciphertexts generated by multiple data owners within one collective key domain.

4 Distributed Setup for Multi-Party SSE-Enabled Threshold CKKS

This section instantiates the distributed setup introduced in Section 3.2. We present the concrete protocols for generating the collective dense and sparse secrets, converting their final Boolean representations into RNS-consistent arithmetic shares, generating the required CKKS and SSE key material, and handing the long-term dense-secret shares to the participating parties.

Party-Contributed Randomness. For secret generation, each party P_m contributes a private uniform random bit string ρ_m of a common public length through Boolean sharing. It samples an independent uniform mask $\rho_m^{(0)}$ of the same length, sets $\rho_m^{(1)} = \rho_m \oplus \rho_m^{(0)}$, and sends $\rho_m^{(c)}$ to C_c. Each auxiliary server locally XORs the received shares: $\rho^{(c)} = \bigoplus_{m=1}^{M} \rho_m^{(c)}, c \in \{0,1\}$. Under the randomness assumption in Section B, the auxiliary servers use disjoint portions of this shared bit string as independent randomness for the dense-secret and sparse-secret samplers.

4.1 Distributed Dense Secret Generation

The protocol constructs the secret $\mathbf{s}_d = (s_{d,0}, \ldots, s_{d,N-1})$, where $s_{d,u} \in \{-1, 0, 1\}$, under Boolean sharing before realizing it over the CKKS RNS basis. Algorithm 1 summarizes the complete procedure in Appendix C.1.

Candidate Generation. Given a candidate budget T_d, the auxiliary servers obtain the Boolean-shared candidate pairs $\left\{\left([\![a_k]\!]_{\mathsf{B}}^{\mathcal{C}}, [\![b_k]\!]_{\mathsf{B}}^{\mathcal{C}}\right)\right\}_{k=1}^{T_d}$ in parallel. Each pair encodes $(0,0) \mapsto 0, (1,0) \mapsto +1, (0,1) \mapsto -1$, while $(1,1)$ is invalid. For all $k \in [T_d]$, the auxiliary servers secure compute $[\![v_{d,k}]\!]_{\mathsf{B}}^{\mathcal{C}} = \neg\mathsf{AND}_{\mathsf{B}}\left([\![a_k]\!]_{\mathsf{B}}^{\mathcal{C}}, [\![b_k]\!]_{\mathsf{B}}^{\mathcal{C}}\right)$ in parallel.

Validity Filtering. The validity vector $\mathbf{v}_d = (v_{d,1}, \ldots, v_{d,T_d})$ is opened in one round. Let $\Pi_d = (k_1, \ldots, k_{|\Pi_d|})$ denote the ordered indices satisfying $v_{d,k} = 1$. If $|\Pi_d| < N$, the protocol returns $\perp$. Otherwise, the first N valid candidates are selected as $\pi(u) = k_{u+1}$.

Secret Assembly. For each coefficient u, the selected pair defines $[\![P_{d,u}]\!]_{\mathsf{B}}^{\mathcal{C}} = [\![a_{\pi(u)}]\!]_{\mathsf{B}}^{\mathcal{C}}$, and $[\![M_{d,u}]\!]_{\mathsf{B}}^{\mathcal{C}} = [\![b_{\pi(u)}]\!]_{\mathsf{B}}^{\mathcal{C}}$. Collecting all coefficients yields $[\![\mathbf{P}_d]\!]_{\mathsf{B}}^{\mathcal{C}}$, and $[\![\mathbf{M}_d]\!]_{\mathsf{B}}^{\mathcal{C}}$. These Boolean shares are subsequently converted into CKKS-compatible RNS shares using the protocol in Section 4.3.

4.2 Distributed Sparse Secret Generation

The protocol constructs the secret $\mathbf{s}_s = (s_{s,0}, \ldots, s_{s,N-1})$ and $\mathrm{HW}(\mathbf{s}_s) = h$, where exactly $h/2$ coefficients are assigned to $+1$ and the remaining $h/2$ coefficients are assigned to -1. Algorithm 3 in Appendix C.3 summarizes the complete procedure.

Candidate Support Generation. Let $N = 2^n$. Using the party-contributed randomness defined above, the auxiliary servers instantiate B_s candidate support batches in parallel. For each $k \in [B_s]$ and $t \in [h]$, the t-th support position is represented by $[\![j_{k,t}]\!]_{\mathsf{B}}^{\mathcal{C}} = \left([\![j_{k,t,0}]\!]_{\mathsf{B}}^{\mathcal{C}}, \ldots, [\![j_{k,t,n-1}]\!]_{\mathsf{B}}^{\mathcal{C}}\right)$. Each batch therefore contains $\left\{[\![j_{k,t}]\!]_{\mathsf{B}}^{\mathcal{C}}\right\}_{t=1}^{h}$.

Collision Filtering. For all $k \in [B_s]$ and $1 \le t < t' \le h$, the auxiliary servers compute $[\![e_{k,t,t'}]\!]_{\mathsf{B}}^{\mathcal{C}} = \mathsf{EQ}([\![j_{k,t}]\!]_{\mathsf{B}}^{\mathcal{C}}, [\![j_{k,t'}]\!]_{\mathsf{B}}^{\mathcal{C}})$ in parallel, then obtain $[\![v_{s,k}]\!]_{\mathsf{B}}^{\mathcal{C}} = \mathsf{AND}(\{\neg[\![e_{k,t,t'}]\!]_{\mathsf{B}}^{\mathcal{C}}\}_{t<t'})$. Both the bitwise reduction within EQ and the outer non-collision reduction use balanced Boolean AND trees, with same-level gates batched across independent instances for logarithmic interaction depth. The batch-validity vector $\mathbf{v}_s = (v_{s,1}, \ldots, v_{s,B_s})$ is then opened. If no valid batch exists, the protocol aborts with $\perp$. Otherwise, they select $k^\star = \min\{k \in [B_s] \mid v_{s,k} = 1\}$ for private expansion.

Private Expansion. For each selected index $[\![j_t]\!]_{\mathsf{B}}^{\mathcal{C}}$, the auxiliary servers compute $[\![\mathbf{e}_t]\!]_{\mathsf{B}}^{\mathcal{C}} = \mathsf{PrivateExpand}\left([\![j_t]\!]_{\mathsf{B}}^{\mathcal{C}}\right)$, where $[\![\mathbf{e}_t]\!]_{\mathsf{B}}^{\mathcal{C}}$ is an N-dimensional one-hot vector with its nonzero entry at position j_t. The expansion has $n = \log_2 N$ layers. At layer ℓ, all $(t, m) \in [h] \times \{0, \ldots, 2^\ell - 1\}$ nodes are evaluated in parallel, with their Boolean AND gates batched within the same interaction layer. The layers are processed sequentially, yielding Boolean depth $\log_2 N$. Algorithm 2 gives the procedure in Appendix C.2

Secret Assembly. The signs are assigned according to the sampling order. The first $h/2$ support positions form the positive indicator vector $[\![\mathbf{P}_s]\!]_{\mathsf{B}}^{\mathcal{C}} = \bigoplus_{t=1}^{h/2} [\![\mathbf{e}_t]\!]_{\mathsf{B}}^{\mathcal{C}}$, and the remaining $h/2$ the negative one: $[\![\mathbf{M}_s]\!]_{\mathsf{B}}^{\mathcal{C}} = \bigoplus_{t=h/2+1}^{h} [\![\mathbf{e}_t]\!]_{\mathsf{B}}^{\mathcal{C}}$. The protocol outputs $[\![\mathbf{P}_s]\!]_{\mathsf{B}}^{\mathcal{C}}$, and $[\![\mathbf{M}_s]\!]_{\mathsf{B}}^{\mathcal{C}}$, which are subsequently converted into CKKS-compatible RNS shares using the protocol in Section 4.3.

4.3 RNS-Consistent Secret Conversion

For each $x \in \{d, s\}$, the secret-generation protocol produces $[\![\mathbf{P}_x]\!]_{\mathsf{B}}^{\mathcal{C}}$ and $[\![\mathbf{M}_x]\!]_{\mathsf{B}}^{\mathcal{C}}$. Subsequent CKKS key generation instead requires arithmetic shares over the RNS basis $\mathcal{B} = \{q_0, \ldots, q_{L_{\mathcal{B}}-1}\}$. We retain all intermediate states in the Boolean domain and convert only the two final indicator vectors.

RNS-Consistent Correlated Bits. The preprocessing shown in Appendix C.4 generates, for $r = r_{\mathsf{B}}^{(0)} \oplus r_{\mathsf{B}}^{(1)}$, the correlated sharing $([\![r]\!]_{\mathsf{B}}^{\mathcal{C}}, \{[\![r]\!]_{q_\ell}^{\mathcal{C}}\}_{\ell=0}^{L_{\mathcal{B}}-1})$. Server C_c holds $r_{\mathsf{B}}^{(c)}$ and $\{r_\ell^{(c)}\}_{\ell=0}^{L_{\mathcal{B}}-1}$, where $r_{\mathsf{B}}^{(0)} \oplus r_{\mathsf{B}}^{(1)} = r$ and $r_\ell^{(0)} + r_\ell^{(1)} \equiv r \pmod{q_\ell}$. Thus, every arithmetic sharing represents the same underlying bit as the Boolean sharing.

Boolean-to-RNS Conversion. Given $[\![z]\!]_{\mathsf{B}}^{\mathcal{C}} = (z_{\mathsf{B}}^{(0)}, z_{\mathsf{B}}^{(1)})$, the servers consume one fresh correlated bit and open only $\delta = \mathsf{Open}_{\mathsf{B}}([\![z]\!]_{\mathsf{B}}^{\mathcal{C}} \oplus [\![r]\!]_{\mathsf{B}}^{\mathcal{C}}) = z \oplus r$. For every RNS limb, they locally compute $z_\ell^{(c)} = (1 - 2\delta) r_\ell^{(c)} + \mathbf{1}[c = 0]\delta \pmod{q_\ell}$, where $c \in \{0, 1\}$. Consequently,

$$z_\ell^{(0)} + z_\ell^{(1)} \equiv \delta + (1 - 2\delta)\left(r_\ell^{(0)} + r_\ell^{(1)}\right) \equiv \delta + (1 - 2\delta) r \equiv z \pmod{q_\ell}. \tag{4}$$

Table 2: Asymptotic and exact setup costs.

Component	Dominant work	Online rounds	Main scaling
Dense-secret sampling	T_d Boolean ANDs	2	$\mathcal{O}(T_d)$
Sparse-support validation	$B_s(J_h n - 1)$ Boolean ANDs	$\lceil \log_2 n \rceil + \lceil \log_2 J_h \rceil + 1$	$\mathcal{O}(B_s h^2 \log N)$
Private expansion	$h(N-1)$ Boolean ANDs	n	$\mathcal{O}(hN)$
B2RNS conversion	$4N$ openings, $4NL_{\mathcal{B}}$ RNS ops	1	$\mathcal{O}(NL_{\mathcal{B}})$
SSE switching keys	$O(D_g N(L_{qp} + L_{QP}))$	1	RNS/key parameters
Ownership handoff	$O(MN \log Q)$ bits	1	$\mathcal{O}(MN)$

The protocol never reconstructs or opens z. Algorithm 4 in Appendix C.5 summarizes the procedure.

Batch Secret Conversion. For each $x \in \{d, s\}$, conversion is applied only to $[\![\mathbf{P}_x]\!]^{\mathcal{C}}_{\mathsf{B}}$ and $[\![\mathbf{M}_x]\!]^{\mathcal{C}}_{\mathsf{B}}$, so one secret requires exactly $2N$ Boolean-to-RNS conversions. Since these conversions are independent and use fresh RNS-consistent correlated bits, they can be executed in parallel within a single interaction round.

After conversion, each server locally computes $[\![s_x]\!]^{\mathcal{C}}_{\mathcal{B}} = [\![\mathbf{P}_x]\!]^{\mathcal{C}}_{\mathcal{B}} - [\![\mathbf{M}_x]\!]^{\mathcal{C}}_{\mathcal{B}}$, where the subtraction is performed component-wise over the RNS basis. The resulting dense-secret and sparse-secret sharings $[\![s_d]\!]^{\mathcal{C}}_{\mathcal{B}}$ and $[\![s_s]\!]^{\mathcal{C}}_{\mathcal{B}}$ are then provided to the distributed CKKS key-generation procedure.

4.4 SSE-Enabled Threshold CKKS Key Material Generation

After the RNS-consistent conversion, the auxiliary servers hold temporary arithmetic shares of the collective dense and sparse secrets over the RNS bases required by the adopted threshold-CKKS and SSE backends. These shares are used directly for distributed key generation without reconstructing either s_d or s_s. The generation details of these keys are provided in the Appendix C.6.

Dense-Domain Key Material. Using the RNS-consistent shares of s_d, the auxiliary servers invoke the adopted threshold-CKKS key-generation procedures Mouchet et al. (2021) to construct the collective public key pk and the conventional evaluation-key material $(\mathsf{evk}_{\mathrm{rlin}}, \mathsf{evk}_{\mathrm{rot}})$ required by secure transformer inference: $\mathcal{K}_d = (pk, \mathsf{evk}_{\mathrm{rlin}}, \mathsf{evk}_{\mathrm{rot}})$.

Cross-Secret Switching Material. SSE additionally requires dense-to-sparse and sparse-to-dense switching keys, both generated locally from secret shares via the affine-linear CKKS switching-key relation. Using the corresponding RNS shares $\{[\![s_d]\!]^{\mathcal{C}}_{\mathcal{B}_{qp}}, [\![s_s]\!]^{\mathcal{C}}_{\mathcal{B}_{qp}}\}$ and $\{[\![s_s]\!]^{\mathcal{C}}_{\mathcal{B}_{QP}}, [\![s_d]\!]^{\mathcal{C}}_{\mathcal{B}_{QP}}\}$, the servers generate $\mathcal{K}^{qp}_{d\to s}$ and $\mathcal{K}^{QP}_{s\to d}$, respectively.

Synchronized Decryption State. The parties additionally establish the private PRF keys and masking state required for synchronized threshold decryption Colin de Verdière et al. (2026), following the original setup procedure unchanged. This state is retained locally and is not included in $\mathcal{K}_{\mathrm{pub}}$.

Dense-Secret Ownership Handoff. After $\mathcal{K}_{\mathrm{pub}}$ is fixed, the dense secret s_d is transferred from the temporary two-server sharing domain to the long-term M-party decryption domain. The auxiliary servers use a resharing protocol to redistribute the shares of s_d into M shares and send each share to the corresponding participant P_m. No long-term share of s_s is required, since the adopted SSE path returns ciphertexts to the dense domain before threshold decryption. After the handoff, the auxiliary servers no longer participate in online inference.

Complexity analysis and security are analyzed in Sections 4.5 and E.

4.5 Complexity Analysis

Table 2 summarizes the computation and online interaction costs of the distributed setup. Detailed operation counts and sampling-failure derivations are deferred to Appendix F. Let $n = \log_2 N$, $J_h = \binom{h}{2}$, and $L_{\mathcal{B}} = |\mathcal{B}|$. The reported rounds count interactions between C_0 and C_1 after receiving the parties' randomness contributions.

5 ONLINE MULTI-USER INFERENCE

After the distributed setup, all parties use the collective public key pk, while the inference server obtains the public evaluation material $\mathcal{K}_{\text{pub}}$. Under the interleaved layout of Zhang et al. (2026), we pre-assign each concurrently processed party P_m a set of positions Ω_m. P_m embeds its input into these positions and independently encrypts $ct_m = \mathsf{Enc}_{pk}(\mathsf{Encode}(\mathsf{Embed}_m(x_m; \Omega_m)))$. Hence, ciphertexts from different parties share the same CKKS key domain, enabling MOAI-style inference Zhang et al. (2026) over the entire input batch and producing a batched ciphertext output. After inference, the server distributes the complete batched ciphertext ct_y to all parties. Following the pre-agreed interleaved layout, each party locally reorganizes the ciphertext and extracts a recipient-specific ciphertext for each P_m. The extracted ciphertext is then processed using the synchronized threshold-decryption protocol Colin de Verdière et al. (2026). Each party computes and masks its partial-decryption share as prescribed, and sends it to the corresponding P_m, which locally aggregates the shares to recover its output. The detail process is shown in Appendix D

6 EVALUATION

We evaluate the framework at both protocol and system levels through the following research questions, with the main results reported here and extended experiments provided in the Appendix. G.

- RQ1: Do the proposed protocols generate the dense and fixed-weight sparse secrets required by SSE while preserving target distributions, structural constraints, and cross-limb RNS consistency?
- RQ2: How efficient is the proposed SSE-enabled setup, and what benefits does Boolean-first delayed RNS realization provide?
- RQ3: How does the two-server-assisted setup scale with participants and cryptographic parameters, and how are its costs distributed?
- RQ4: Can the proposed framework support efficient and correct encrypted transformer inference for multiple independent users?

The security of the proposed construction is established analytically in Appendix E. The following experiments evaluate distribution correctness, efficiency, scalability, and system-level performance.

6.1 EXPERIMENTAL SETUP

Implementation. We implement the distributed setup framework in *C++* using *libOTe*[1]. Boolean-domain secure computation is instantiated with GMW, while SilentOT provides the correlated randomness required by Boolean computation and RNS-consistent conversion. RNS arithmetic and threshold CKKS operations are implemented using *SEAL*[2] (CPU) and *Phantom*[3] (GPU). The encrypted transformer inference follows the MOAI Zhang et al. (2026) implementation [4].

The distributed setup experiments run on servers equipped with an AMD Ryzen 9 9950X CPU and 96 GB RAM. We evaluate LLaMA3-8B inference on user requests with a maximum sequence length of 512 tokens. At $N = 2^{16}$, the 32,768 CKKS slots are partitioned into 64 groups of 512 positions, with up to 511 positions used for input tokens and one reserved for output. We use SST-2, QNLI, and RTE from GLUE benchmark[5], with a separately LoRA-fine-tuned LLaMA3-8B model for each task. The experiments run on a server equipped with an AMD EPYC 9K84 CPU, 512 GB RAM, and four NVIDIA H20 GPUs connected via NVLink. Our implementation is available in the repository[6].

Cryptographic Parameters. We evaluate CKKS configurations with $N = 2^{16}$ and $h = 32$. The target RNS basis is denoted by $\mathcal{B} = q_0, \ldots, q_{L_{\mathcal{B}}-1}$ with $L_{\mathcal{B}} = 40$. We target 128-bit security with $|QP| < 1747$ bits Bossuat et al. (2024) and $|qp| < 121$ bits Bossuat et al. (2022). Threshold decryption uses Gaussian smudging noise with $\sigma_{\text{smdg}} = 2^{28}$.

[1] https://github.com/osu-crypto/libOTe.
[2] https://github.com/microsoft/SEAL.
[3] https://github.com/encryptorion-lab/phantom-fhe.
[4] https://github.com/dtc2025ag/MOAI_GPU
[5] https://gluebenchmark.com/
[6] https://anonymous.4open.science/r/DKG-Core-EFC5

Table 3: Distributional correctness of the generated collective secrets.

(a) Dense secret

Method	$\widehat{p}_{-1}$	$\widehat{p}_0$	$\widehat{p}_{+1}$	$\widehat{p}_\perp$	D_{TV}
Target	$1/3$	$1/3$	$1/3$	0	0
Local Agg.	0.11582	0.12047	0.11522	0.64850	0.64850
SEAL	0.33379	0.33380	0.33241	0	0.00093
Ours	**0.33451**	**0.33338**	**0.33211**	**0**	**0.00123**

(b) Sparse secret

(N, h)	$D_{\mathrm{TV}}^{\mathrm{bin}}$	95% interval	p_{batch} (E/A)	B_{40}
$(2^{14}, 16)$	0.01079	[0.01028, 0.01227]	0.99236/0.99270	6
$(2^{15}, 32)$	0.00786	[0.00717, 0.00869]	0.98585/0.98497	7
$(2^{16}, 32)$	0.00866	[0.00719, 0.00869]	0.99172/0.99246	6
$(2^{16}, 64)$	0.00583	[0.00518, 0.00619]	0.97083/0.96970	8

6.2 RQ1: Distribution-Preserving Secret Generation

We evaluate whether the implementation preserves the prescribed dense and balanced fixed-weight sparse collective secrets.

Dense Secret. With $M = 16$, we reconstructed 524,288 coefficients for validationmeasure deviation from the target uniform ternary distribution using $D_{\mathrm{TV}}^{\mathrm{coef}} = \frac{1}{2}\sum_{z\in\{-1,0,+1,\perp\}} |\widehat{p}_z - p_z|$, where $\widehat{p}_z$ is the empirical coefficient frequency in category z, and $\perp$ groups all non-ternary values. We use local aggregation and SEAL as distributional references. Table 3a reports $D_{\mathrm{TV}}^{\mathrm{coef}} = 0.00123$ for our protocol and 0.00093 for SEAL, with no out-of-ternary coefficients, supporting agreement with the target distribution. In contrast, local aggregation places 0.64850 of its mass outside the ternary.

Sparse Secret. For each configuration in Table 3b, all full-protocol runs passed checks of fixed Hamming weight, sign balance, and cross-limb RNS consistency. Sampler-level repetitions evaluated positional support uniformity using $D_{\mathrm{TV}}^{\mathrm{bin}} = \frac{1}{2}\sum_{j=1}^{256} \left|\widehat{\pi}_j - \frac{1}{256}\right|$, where $\widehat{\pi}_j$ is the fraction of accepted support positions in the jth of 256 equal-size bins. We also compared empirical batch-acceptance rates with the analytical probability $p_{\mathrm{batch}} = \prod_{k=0}^{h-1}\left(1 - \frac{k}{N}\right)$. The observed $D_{\mathrm{TV}}^{\mathrm{bin}}$ values range from 0.00583 to 0.01079, all within the matched 95% ideal-sampling reference intervals. Empirical acceptance rates deviate from analytical predictions by at most 0.00113, with $B_{40} = 6$–8 candidate batches sufficient to bound the failure probability by 2^{-40}. These results are consistent with the expected binned positional uniformity and rejection-sampling behavior.

6.3 RQ2: SSE-Enabled Distributed Setup Efficiency

We evaluate the concrete cost of the SSE-enabled distributed setup. The benefits of Boolean-first generation and delayed RNS realization are further evaluated in Appendix G.1.

Setup Cost Breakdown. For $M = 16$, we profile the proposed two-server-assisted setup for a key set comprising one dense-secret public key, two cross-secret switching keys, one rotation (galois) key, and relinearization material. Table 4 shows that dense-secret and sparse-secret generation is lightweight, whereas generating the sparse-to-dense switching key, galois key, and relinearization material dominates latency and communication. The filtered online setup interval is $20.226\,\mathrm{s}$, with input-independent preparation taking $14.404\,\mathrm{s}$ separately. For additional context, Table 5 places our standalone sparse-secret generation measurements alongside the threshold-CKKS DKG Min et al. (2025).

Table 4: Setup cost breakdown at $N = 2^{16}$, $h = 32$, $L_{\mathcal{B}} = 40$, and $M = 16$.

Component / Scope	Time (s)	Comm. (MB)
Dense secret	0.121	44.139
Sparse secret	0.227	45.097
Threshold PK	0.197	62.915
D→S KSK	0.182	62.915
S→D KSK	6.195	2453.676
Galois key ($K = 1$)	6.242	2453.676
Relin. material	6.585	2495.619
Dense-share handoff	0.506	671.089
Input-independent preparation	14.404	–
Filtered online setup interval	20.226	–

6.4 RQ3: Scalability of the Two-Server-Assisted Setup

We evaluate setup scalability as M grows, focusing on per-party costs and the distribution of computation and communication costs across parties and auxiliary servers, with cryptographic parameter-scaling results reported in Appendix G.2.3.

Party-Count Scaling. We vary $M \in \{2, 4, 8, 16, 32, 64\}$ under LAN and WAN and evaluate setup scalability from a per-party perspective. As shown in Table 6, each party in the two-

Table 5: Comparison of sparse-secret generation methods.

Method	Parties	Hardware	Latency.
DKG Min et al. (2025)	32	GPU	1.480s
DKG Min et al. (2025)	256	GPU	1.640s
DKG Min et al. (2025)	1024	GPU	1.790s
Ours ($M = 32$)	32	CPU	0.222s
Ours ($M = 256$)	256	CPU	0.225s
Ours ($M = 1024$)	1024	CPU	0.283s

Table 6: Party-count scaling with Direct-MPC reference.

Parties M	CPU LAN/WAN (ms)	Setup LAN/WAN (s)	Server-assisted In/Handoff (MB)	Direct-MPC Online/Prep. (MB)
2	39.654/43.355	4.512/22.179	0.531/41.943	1.217/167.772
4	39.457/42.790	5.203/29.073	0.531/41.943	3.652/503.316
8	42.190/47.109	6.797/42.895	0.531/41.943	8.522/1174.405
16	46.587/50.190	8.786/58.225	0.531/41.943	18.262/2516.582
32	48.206/56.088	12.386/85.585	0.531/41.943	37.742/5200.937
64	51.443/52.920	19.961/144.976	0.531/41.943	76.701/10569.646

Table 7: Inference correctness.

Dataset	PT acc	MU acc	PT-MU Flips	SU-MU Flips
SST-2	0.976	0.975	4/1821	0/1821
QNLI	0.956	0.956	14/5463	0/5463
RTE	0.915	0.912	21/3000	0/3000

Table 8: Inference scalability.

U	Separate (s)	MU interface (s)	Multi-user (s)	Speedup
1	426378.142	0.415	426172.284	$1.000\times$
4	1704173.540	0.804	426603.502	$3.995\times$
16	6818527.740	2.746	426210.629	$\mathbf{15.998}\times$

server setup incurs a fixed 0.531 $\mathrm{MB}/41.943\mathrm{MB}$ for randomness input/dense-share handoff, independent of M. In contrast, the maximum-party Direct-MPC online communication grows from $1.217\mathrm{MB}$ to $76.701\mathrm{MB}$, while the estimated per-party B2RNS preprocessing payload increases from $167.772\mathrm{MB}$ to $10.570\mathrm{GB}$. Setup latency increases by $4.42\times/6.54\times$ under LAN/WAN, while CPU time shows only modest growth with M and, being contention-sensitive, may overstate typical party cost. Appendix G.2.1 details the B2RNS preprocessing estimate, while Appendix G.2.2 attributes the latency increase primarily to auxiliary-server processing and dense-share handoff.

6.5 RQ4: Independent-User Interleaved Inference

RQ4 evaluates whether independent users can share a single interleaved encrypted inference while preserving per-user correctness. Separate execution is measured directly, while multi-user latency combines the measured transformer runtime and interface cost.

Independent-User Correctness. Table 7 evaluates task-level correctness across plaintext (PT), single-user encrypted (SU), and multi-user encrypted (MU) inference. MU inference incurs only 4/1821, 14/5463, and 21/3000 prediction disagreements relative to PT on SST-2, QNLI, and RTE, respectively, while preserving comparable task accuracy. In contrast, SU and MU predictions agree on every evaluated sample, demonstrating that multi-user consolidation preserves the prediction behavior of single-user encrypted inference without introducing additional decision-level errors.

Cross-User Consolidation. Each active user contributes one distinct request. Separate execution evaluates users individually, whereas multi-user execution consolidates them into one shared batch. Latency measures one complete interleaved evaluation without per-request amortization. Table 8 shows that cross-user consolidation achieves a $15.998\times$ speedup at $U = 16$, while the multi-user interface adds only 2.746 s, or 0.00064% of the inference runtime. The gain comes from sharing otherwise underutilized packing capacity across independent users, rather than accelerating the HE operators or enlarging the packing capacity.

7 Conclusion and Future Work

We presented an SSE-enabled threshold-CKKS framework for multi-user secure transformer inference. It enables independently encrypted requests from multiple users to share one collective key domain and one interleaved transformer evaluation, while preserving user-specific output regions and threshold-controlled release. In the evaluation, consolidating 16 users reduces projected service time by $15.998\times$ with 0.00064% overhead while preserving prediction accuracy. As a path forward, we believe two promising avenues warrant further investigation. One direction is to strengthen security against malicious adversaries through verifiable setup and maliciously secure threshold decryption. Additionally, future work could support dynamic users through efficient admission and departure, elastic packing, and incremental key reconfiguration.

AI USE STATEMENT

In this work, we used generative AI tools for language polishing, including spelling and grammar checking, and for assisting with code organization and environment setup. We also used generative AI collaboratively to help organize and refine the presentation of the proofs in Section E. We did not use generative AI to generate experimental data or results. All AI-assisted work was reviewed by the authors: the proof arguments and technical claims were independently checked, and AI-assisted code and environment configurations were tested for correctness. We take responsibility for the final content of this work, including text, claims, code, and other artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work does not involve human subjects, private or sensitive data, or newly collected datasets. All datasets used in the evaluation are publicly available. We are not aware of any additional ethical concerns associated with this work.

REPRODUCIBILITY STATEMENT

Our implementation code is publicly available through the anonymous GitHub repository provided in the submission materials. All datasets used in our experiments are publicly available. Details of the parameter settings, libraries, models, datasets, and hardware configurations are provided in Section 6.1. The DKG runtimes reported in Table 5 are taken from the original paper and are included only as a reference due to differences in system models and experimental environments. The inference times in Table 8 were measured using the MOAI source code adapted to the SSE setting. For this experiment, the repository provides the SSE-mode bootstrapping implementation for integration with the original MOAI codebase, in addition to the implementations of the other components evaluated in this work. With these resources, all results reported in this paper are reproducible.

REFERENCES

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

Ahmad Al Badawi, Andreea Alexandru, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Carlo Pascoe, Yuriy Polyakov, Ian Quah, Saraswathy R.V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, Vinod Vaikuntanathan, and Vincent Zucca. OpenFHE: Open-source fully homomorphic encryption library. Cryptology ePrint Archive, Paper 2022/915, 2022. URL https://eprint.iacr.org/2022/915.

Dan Boneh, Rosario Gennaro, Steven Goldfeder, Aayush Jain, Sam Kim, Peter M. R. Rasmussen, and Amit Sahai. Threshold cryptosystems from threshold fully homomorphic encryption. In Hovav Shacham and Alexandra Boldyreva (eds.), *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part I*, volume 10991 of *Lecture Notes in Computer Science*, pp. 565–596. Springer, 2018. doi: 10.1007/978-3-319-96884-1_19. URL https://doi.org/10.1007/978-3-319-96884-1_19.

Jean-Philippe Bossuat, Juan Ramón Troncoso-Pastoriza, and Jean-Pierre Hubaux. Bootstrapping for approximate homomorphic encryption with negligible failure-probability by using sparse-secret encapsulation. In *Applied Cryptography and Network Security*, pp. 521–541. Springer, 2022. doi: 10.1007/978-3-031-09234-3_26. URL https://eprint.iacr.org/2022/024.

Jean-Philippe Bossuat, Rosario Cammarota, Ilaria Chillotti, Benjamin R. Curtis, Wei Dai, Huijing Gong, Erin Hales, Duhyeong Kim, Bryan Kumara, Changmin Lee, Xianhui Lu, Carsten Maple, Alberto Pedrouzo-Ulloa, Rachel Player, Yuriy Polyakov, Luis Antonio Ruiz Lopez,

Yongsoo Song, and Donggeon Yhee. Security guidelines for implementing homomorphic encryption. *IACR Commun. Cryptol.*, 1(4):26, 2024. doi: 10.62056/ANXRA69P1. URL https://doi.org/10.62056/anxra69p1.

François Colin de Verdière, Alain Passelègue, and Damien Stehlé. On threshold fully homomorphic encryption with synchronized decryptors. Cryptology ePrint Archive, Paper 2026/031, 2026. URL https://eprint.iacr.org/2026/031.

Zhenkai Hu, Haofei Liang, Xiao Wang, Xiang Xie, Kang Yang, Yu Yu, and Wenhao Zhang. Ajax: Fast threshold fully homomorphic encryption without noise flooding. *IACR Cryptol. ePrint Arch.*, 2025:1834, 2025. URL https://eprint.iacr.org/2025/1834.

Yongwoo Lee, Daniele Micciancio, Andrey Kim, Rakyong Choi, Maxim Deryabin, Jieun Eom, and Donghoon Yoo. Efficient FHEW bootstrapping with small evaluation keys, and applications to threshold homomorphic encryption. In Carmit Hazay and Martijn Stam (eds.), *Advances in Cryptology - EUROCRYPT 2023 - 42nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Lyon, France, April 23-27, 2023, Proceedings, Part III*, volume 14006 of *Lecture Notes in Computer Science*, pp. 227–256. Springer, 2023. doi: 10.1007/978-3-031-30620-4_8. URL https://doi.org/10.1007/978-3-031-30620-4_8.

Baiyu Li and Daniele Micciancio. On the security of homomorphic encryption on approximate numbers. In Anne Canteaut and François-Xavier Standaert (eds.), *Advances in Cryptology - EUROCRYPT 2021 - 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17-21, 2021, Proceedings, Part I*, volume 12696 of *Lecture Notes in Computer Science*, pp. 648–677. Springer, 2021. doi: 10.1007/978-3-030-77870-5_23. URL https://doi.org/10.1007/978-3-030-77870-5_23.

Zhengyi Li, Kang Yang, Jin Tan, Wen-jie Lu, Haoqi Wu, Xiao Wang, Yu Yu, Derun Zhao, Yancheng Zheng, Minyi Guo, and Jingwen Leng. Nimbus: Secure and efficient two-party inference for transformers. In Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (eds.), *Advances in Neural Information Processing Systems 37: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, 2024. URL http://papers.nips.cc/paper_files/paper/2024/hash/264a9b3ce46abdf572dcfe0401141989-Abstract-Conference.html.

Wen-jie Lu, Zhicong Huang, Zhen Gu, Jingyu Li, Jian Liu, Cheng Hong, Kui Ren, Tao Wei, and Wenguang Chen. BumbleBee: Secure two-party inference framework for large transformers. In *32nd Annual Network and Distributed System Security Symposium, NDSS 2025, San Diego, California, USA, February 24-28, 2025*. The Internet Society, 2025. URL https://www.ndss-symposium.org/ndss-paper/bumblebee-secure-two-party-inference-framework-for-large-transformers/.

Seonhong Min, Guillaume Hanrot, Jai Hyun Park, Alain Passelègue, and Damien Stehlé. Distributed key generation for efficient threshold-CKKS. Cryptology ePrint Archive, Paper 2025/2057, 2025. URL https://eprint.iacr.org/2025/2057.

Jungho Moon, Dongwoo Yoo, Xiaoqian Jiang, and Miran Kim. THOR: secure transformer inference with homomorphic encryption. In Chun-Ying Huang, Jyh-Cheng Chen, Shiuh-Pyng Shieh, David Lie, and Véronique Cortier (eds.), *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security, CCS 2025, Taipei, Taiwan, October 13-17, 2025*, pp. 3765–3779. ACM, 2025. doi: 10.1145/3719027.3765150. URL https://doi.org/10.1145/3719027.3765150.

Christian Mouchet, Juan Ramón Troncoso-Pastoriza, Jean-Philippe Bossuat, and Jean-Pierre Hubaux. Multiparty homomorphic encryption from ring-learning-with-errors. *Proc. Priv. Enhancing Technol.*, 2021(4):291–311, 2021. doi: 10.2478/POPETS-2021-0071. URL https://doi.org/10.2478/popets-2021-0071.

Christian Mouchet, Elliott Bertrand, and Jean-Pierre Hubaux. An efficient threshold access-structure for RLWE-based multiparty homomorphic encryption. *J. Cryptol.*, 36(2):10, 2023. doi: 10.1007/S00145-023-09452-8. URL https://doi.org/10.1007/s00145-023-09452-8.

Qi Pang, Jinhao Zhu, Helen Möllering, Wenting Zheng, and Thomas Schneider. BOLT: privacy-preserving, accurate and efficient inference for transformers. In *IEEE Symposium on Security and Privacy, SP 2024, San Francisco, CA, USA, May 19-23, 2024*, pp. 4753–4771. IEEE, 2024. doi: 10.1109/SP54263.2024.00130. URL https://doi.org/10.1109/SP54263.2024.00130.

Jiawen Zhang, Xinpeng Yang, Lipeng He, Kejia Chen, Wen-jie Lu, Yinghao Wang, Xiaoyang Hou, Jian Liu, Kui Ren, and Xiaohu Yang. Secure transformer inference made non-interactive. In *32nd Annual Network and Distributed System Security Symposium, NDSS 2025, San Diego, California, USA, February 24-28, 2025*. The Internet Society, 2025. URL https://www.ndss-symposium.org/ndss-paper/secure-transformer-inference-made-non-interactive/.

Linru Zhang, Xiangning Wang, Jun Jie Sim, Zhicong Huang, Jiahao Zhong, Huaxiong Wang, Pu Duan, and Kwok Yan Lam. MOAI: Module-optimizing architecture for non-interactive secure transformer inference. In *International Conference on Learning Representations*, volume 2026, pp. 27927–27953, 2026.

A Related Works

Secure transformer inference improves privacy-preserving evaluation, whereas MHE and SSE address distributed key ownership and heterogeneous-secret bootstrapping. We first review secure transformer inference, followed by the cryptographic infrastructure required for multi-party SSE-enabled evaluation.

A.1 Secure Transformer Inference

Secure transformer inference has been studied primarily in client-server settings. Interactive systems such as BOLT Pang et al. (2024), BumbleBee Lu et al. (2025), and Nimbus Li et al. (2024) combine HE and MPC to reduce the cost of matrix multiplication and nonlinear functions, substantially improving two-party inference.

More recent systems pursue non-interactive evaluation using FHE. NEXUS Zhang et al. (2025) minimizes client involvement, THOR Moon et al. (2025) optimizes ciphertext packing, matrix multiplication, and nonlinear evaluation. MOAI Zhang et al. (2026) further improves cross-layer packing, rotations, and interleaved batching to increase inference efficiency.

Existing work largely assumes a single client–server or common-key cryptographic context. As summarized in Table 9, interleaved batching in MOAI Zhang et al. (2026) can amortize evaluation over multiple inputs, but does not establish collective key infrastructure or support user-specific threshold release. Our work bridge this gap with an SSE-compatible threshold-CKKS infrastructure for shared transformer evaluation over independently supplied ciphertexts and synchronized threshold release of per-user outputs.

Table 9: Comparison of secure transformer inference.

Work	Scheme	Mode	Batch	Multi-user
BOLT Pang et al. (2024)	HE-MPC	Int.	–	–
BumbleBee Lu et al. (2025)	HE-MPC	Int.	–	–
Nimbus Li et al. (2024)	HE-MPC	Int.	–	–
NEXUS Zhang et al. (2025)	FHE	N-Int.	–	–
THOR Moon et al. (2025)	FHE	N-Int.	–	–
MOAI Zhang et al. (2026)	FHE	N-Int.	✓	–
Ours	**ThCKKS+SSE**	**N-Int.**	✓	✓

Notes: Int. and N-Int denote interactive and non-interactive evaluation, respectively. Multi-user denotes collective key ownership with user-specific threshold release.

Table 10: Comparison of collective setup and SSE capabilities.

Work	Setup	Dense	Sparse	SSE	Release
MHE Mouchet et al. (2021)	✓	–	–	–	–
TMHE Mouchet et al. (2023)	✓	–	–	–	–
OpenFHE Badawi et al. (2022)	✓	–	–	–	–
Lee et al. (2023)	✓	–	–	–	–
Ajax Hu et al. (2025)	✓	–	–	–	–
Min et al. (2025)	✓	–	✓	✓	–
SSE Bossuat et al. (2022)	–	✓	✓	✓	–
SynDec. Colin de Verdière et al. (2026)	–	–	–	–	✓
Ours	✓	✓	✓	✓	✓

A.2 Multi-Party Homomorphic Encryption and Sparse-Secret Encapsulation

Multi-Party Homomorphic Encryption (MHE) distributes decryption authority through collective keys formed from independently sampled local shares. Mouchet et al. (2021) establish this framework, their threshold extension Mouchet et al. (2023) adds flexible access structures through share re-sharing, and OpenFHE Badawi et al. (2022) provides practical implementations. However, summing local ternary shares generally yields a convolved rather than a prescribed collective distribution.

SSE Bossuat et al. (2022) combines a dense CKKS secret for regular evaluation with a temporary sparse secret for efficient bootstrapping, connected by dense-sparse switching material. It does not address distributed generation under collective ownership. Lee et al. (2023) support arbitrary aggregate-key distributions in FHEW bootstrapping, while Ajax Hu et al. (2025) jointly samples coefficient-wise binary or non-uniform ternary FHEW keys. Except for Min et al. (2025), none of the cited schemes directly generates either the uniform dense ternary or fixed-weight sparse CKKS secret used in our SSE instantiation. Min et al. (2025) first establishes an initial CKKS key infrastructure using the distributed key-generation protocol of Mouchet et al. (2021), then homomorphically generates additive shares of the sparse and less-sparse secrets through bootstrapping. These shares are subsequently used to establish the target key infrastructure. Since generating the required public and evaluation keys incurs multiparty computation and communication that grow quadratically with the number of parties, key generation becomes a major scalability bottleneck. This overhead is further amplified because both the initial and target key infrastructures must be established.

Separately, Colin de Verdière et al. Colin de Verdière et al. (2026) secure synchronized threshold decryption using PRF-masked partial shares. This mechanism addresses secure output release rather than the generation of the underlying secret distributions. Table 10 summarizes the complementary setup, SSE, and release capabilities of these works.

As shown in Table 10, existing MHE distributes key generation without jointly generating the two secret distributions required by SSE, whereas existing SSE assumes that the corresponding secret domains and switching material are already available. Our setup bridges these directions by directly constructing both collective secrets and deriving RNS-consistent threshold-CKKS and SSE key material without reconstructing either. We adopt synchronized threshold decryption Colin de Verdière et al. (2026) as the output-release backend.

B Threat Model

As shown in Fig. 1, C_0 and C_1 participate only during setup, whereas $\mathcal{S}$ participates only in online inference. Let $[M] = \{1, \ldots, M\}$. We consider a static semi-honest adversary that corrupts a fixed party set $\mathcal{I} \subsetneq [M]$, the inference server $\mathcal{S}$, and optionally one auxiliary server $C_b \in \{C_0, C_1\}$.[7] Corrupted entities follow the prescribed protocols.

[7]The corruption bound follows the access structure in Section 3.1: $|\mathcal{I}| < M$ for our M-of-M instantiation and $|\mathcal{I}| < t$ after t-of-M resharing.

Algorithm 1 Distributed Dense Secret Generation

Input: Secret dimension N, candidate budget T_d
Output: $[\![\mathbf{P}_d]\!]^{\mathcal{C}}_{\mathsf{B}}, [\![\mathbf{M}_d]\!]^{\mathcal{C}}_{\mathsf{B}}$

Collect $\{([\![a_k]\!]^{\mathcal{C}}_{\mathsf{B}}, [\![b_k]\!]^{\mathcal{C}}_{\mathsf{B}})\}_{k=1}^{T_d}$ ▷ parallel
For all $k \in [T_d]$, Compute
    $[\![v_{d,k}]\!] = \neg\mathsf{AND}_{\mathsf{B}}([\![a_k]\!], [\![b_k]\!])$ ▷ parallel
Open $\{v_{d,k}\}_{k=1}^{T_d}$ ▷ batched
if $\sum_{k=1}^{T_d} v_{d,k} < N$ **then**
    return $\perp$
end if
Let $\pi(0), \ldots, \pi(N-1)$ be the first N valid indices
For all $u \in \{0, \ldots, N-1\}$, set
    $[\![P_{d,u}]\!] = [\![a_{\pi(u)}]\!]$, $[\![M_{d,u}]\!] = [\![b_{\pi(u)}]\!]$ ▷ parallel
return $[\![\mathbf{P}_d]\!]^{\mathcal{C}}_{\mathsf{B}}, [\![\mathbf{M}_d]\!]^{\mathcal{C}}_{\mathsf{B}}$

The two auxiliary servers are assumed not to collude, hence at least one of C_0 and C_1 remains honest. For every collective-secret sampling instance, at least one honest contributor supplies fresh randomness unknown to the adversary. We further assume authenticated confidential channels for transmitting secret-dependent shares, including setup and partial-decryption shares. Under these assumptions, we require confidentiality of honest-party inputs and outputs and of the collective secrets s_d and s_s, except for the explicitly permitted leakage $\mathcal{L}_{\mathrm{setup}}$ and $\mathcal{L}_{\mathrm{online}}$ defined in Section E.1.

Malicious deviations, corruption beyond the supported threshold, collusion between C_0 and C_1, and compromise of the assumed secure channels are outside our threat model.

C Detail of Distrubuted Setup

C.1 Algorithm of Distributed Dense Secret Generation

The dense secret generation procedure is shown in Algorithm 1, where $\{([\![a_k]\!]^{\mathcal{C}}_{\mathsf{B}}, [\![b_k]\!]^{\mathcal{C}}_{\mathsf{B}})\}_{k=1}^{T_d}$ are random Boolean shares obtained from the Party-Contributed Randomness phase. The computations $\mathsf{AND}_{\mathsf{B}}([\![a_k]\!], [\![b_k]\!])$ are independent for all $k \in [T_d]$ and can therefore be evaluated in parallel within a single interaction round. Similarly, opening $\{v_{d,k}\}_{k=1}^{T_d}$ can be batched into a single interaction round.

C.2 Detail of Private Expansion

The Private Expansion procedure is shown in Algorithm 2. For each Boolean-shared index $[\![j_t]\!]^{\mathcal{C}}_{\mathsf{B}}$, the auxiliary servers initialize the root as $[\![V_t^{(0)}[0]]\!]^{\mathcal{C}}_{\mathsf{B}} = (1, 0)$, with C_0 holding 1 and C_1 holding 0. Starting from this root, the expansion proceeds level by level. Since the nodes within each level are independent, all right child nodes at level $\ell + 1$ can be computed in parallel within one interaction round as

$$[\![V_t^{(\ell+1)}[2m+1]]\!]^{\mathcal{C}}_{\mathsf{B}} \leftarrow [\![R_{t,m}^{(\ell)}]\!]^{\mathcal{C}}_{\mathsf{B}} \leftarrow \mathsf{AND}_{\mathsf{B}}([\![V_t^{(\ell)}[m]]\!]^{\mathcal{C}}_{\mathsf{B}}, [\![b_{t,n-1-\ell}]\!]^{\mathcal{C}}_{\mathsf{B}}).$$

The corresponding left child nodes are then obtained locally as

$$[\![V_t^{(\ell+1)}[2m]]\!]^{\mathcal{C}}_{\mathsf{B}} \leftarrow [\![V_t^{(\ell)}[m]]\!]^{\mathcal{C}}_{\mathsf{B}} \oplus [\![R_{t,m}^{(\ell)}]\!]^{\mathcal{C}}_{\mathsf{B}}.$$

Moreover, since the Boolean-shared indices are mutually independent, their expansions can be performed in parallel, including the interactive computation of all right child nodes at each level. After all levels are processed, the procedure outputs a Boolean-shared one-hot expansion for each input index. For clarity, Fig. 2 provides a small illustrative example.

C.3 Distributed Sparse Secret Generation

The Sparse Secret Generation procedure is shown in the algorithm, where $\{[\![j_{k,t}]\!]^{\mathcal{C}}_{\mathsf{B}}\}_{k\in[B_s],\, t\in[h]}$ are constructed from the random Boolean shares obtained during the Party-Contributed Randomness

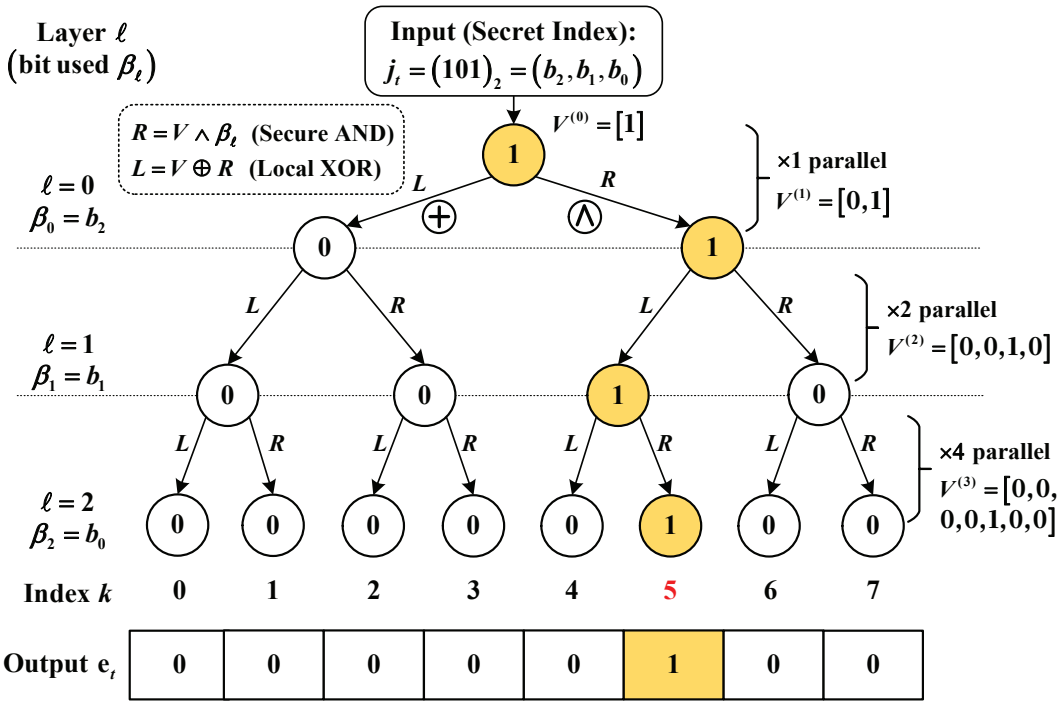

Figure 2: Illustration of PrivateExpand for Boolean one-hot expansion of a selected support index.

Algorithm 2 PrivateExpand: Boolean One-Hot Expansion

Input: Boolean-shared indices $\{[\![j_t]\!]^{\mathcal{C}}_{\mathsf{B}}\}_{t=1}^{h}$, where $j_t = (b_{t,n-1}, \ldots, b_{t,0})$ and $n = \log_2 N$
Output: Boolean-shared one-hot vectors $\{[\![\mathbf{e}_t]\!]^{\mathcal{C}}_{\mathsf{B}}\}_{t=1}^{h}$

Set $[\![V_t^{(0)}[0]]\!]^{\mathcal{C}}_{\mathsf{B}} \leftarrow [\![1]\!]^{\mathcal{C}}_{\mathsf{B}}$ for all $t \in [h]$ ▷ parallel
for $\ell = 0$ to $n-1$ **do**
  for all $(t, m) \in [h] \times \{0, \ldots, 2^\ell - 1\}$ **do** ▷ parallel
    $[\![R_{t,m}^{(\ell)}]\!]^{\mathcal{C}}_{\mathsf{B}} \leftarrow \mathsf{AND}_{\mathsf{B}}([\![V_t^{(\ell)}[m]]\!]^{\mathcal{C}}_{\mathsf{B}}, [\![b_{t,n-1-\ell}]\!]^{\mathcal{C}}_{\mathsf{B}})$
    $[\![V_t^{(\ell+1)}[2m+1]]\!]^{\mathcal{C}}_{\mathsf{B}} \leftarrow [\![R_{t,m}^{(\ell)}]\!]^{\mathcal{C}}_{\mathsf{B}}$
    $[\![V_t^{(\ell+1)}[2m]]\!]^{\mathcal{C}}_{\mathsf{B}} \leftarrow [\![V_t^{(\ell)}[m]]\!]^{\mathcal{C}}_{\mathsf{B}} \oplus [\![R_{t,m}^{(\ell)}]\!]^{\mathcal{C}}_{\mathsf{B}}$
  end for
end for
Set $[\![\mathbf{e}_t]\!]^{\mathcal{C}}_{\mathsf{B}} \leftarrow [\![V_t^{(n)}]\!]^{\mathcal{C}}_{\mathsf{B}}$ for all $t \in [h]$ ▷ parallel
return $\{[\![\mathbf{e}_t]\!]^{\mathcal{C}}_{\mathsf{B}}\}_{t=1}^{h}$

phase. These shares are assembled into bit encodings of candidate positions, with each batch containing h such encodings.

During collision filtering, all pairs of position encodings within each batch are compared as $\mathsf{EQ}\left([\![j_{k,t}]\!]^{\mathcal{C}}_{\mathsf{B}}, [\![j_{k,t'}]\!]^{\mathcal{C}}_{\mathsf{B}}\right)$, where $1 \le t < t' \le h$. EQ first computes the bitwise XOR of the two n-bit encodings and locally negates the resulting bits, followed by an AND reduction over all n equality bits. Using a balanced Boolean AND tree reduces the interaction depth of each equality test from $n-1$ sequential AND rounds to $\lceil \log_2 n \rceil$ rounds. All pairwise equality tests across candidate batches can be evaluated in parallel at each tree level.

For security, each batch reveals only an aggregate validity bit rather than which pair of position encodings collides. Thus, $[\![e_{k,t,t'}]\!]^{\mathcal{C}}_{\mathsf{B}} = \mathsf{EQ}\left([\![j_{k,t}]\!]^{\mathcal{C}}_{\mathsf{B}}, [\![j_{k,t'}]\!]^{\mathcal{C}}_{\mathsf{B}}\right)$. The batch-validity bit is computed as $[\![v_{s,k}]\!]^{\mathcal{C}}_{\mathsf{B}} = \mathsf{AND}_{\mathsf{B}}(\{\neg[\![e_{k,t,t'}]\!]^{\mathcal{C}}_{\mathsf{B}}\}_{1 \le t < t' \le h})$. Since there are $J_h = \binom{h}{2}$ pairwise comparisons, this reduction requires $J_h - 1$ Boolean AND gates but only $\lceil \log_2 J_h \rceil$ interaction rounds when implemented as a balanced Boolean AND tree.

Algorithm 3 Distributed Sparse Secret Generation

Input: Ring dimension $N = 2^n$, even Hamming weight h, candidate budget B_s
Output: $[\![\mathbf{P}_s]\!]^{\mathcal{C}}_{\mathsf{B}}, [\![\mathbf{M}_s]\!]^{\mathcal{C}}_{\mathsf{B}}$

Collect $\{[\![j_{k,t}]\!]^{\mathcal{C}}_{\mathsf{B}}\}_{k\in[B_s],\, t\in[h]}$ ▷ parallel
For all $k \in [B_s]$ and $1 \le t < t' \le h$, compute
    $[\![e_{k,t,t'}]\!]^{\mathcal{C}}_{\mathsf{B}} \leftarrow \mathsf{EQ}\big([\![j_{k,t}]\!]^{\mathcal{C}}_{\mathsf{B}}, [\![j_{k,t'}]\!]^{\mathcal{C}}_{\mathsf{B}}\big)$ ▷ parallel
For all $k \in [B_s]$, compute
    $[\![v_{s,k}]\!]^{\mathcal{C}}_{\mathsf{B}} \leftarrow \mathsf{AND}_{\mathsf{B}}\left(\{\neg[\![e_{k,t,t'}]\!]^{\mathcal{C}}_{\mathsf{B}}\}_{1\le t<t'\le h}\right)$ ▷ batched tree reduction
Open the batch-validity vector $\mathbf{v}_s = (v_{s,1}, \ldots, v_{s,B_s})$
if $\sum_{k=1}^{B_s} v_{s,k} = 0$ **then**
    abort with $\perp$
end if
Select $k^\star \leftarrow \min\{k \in [B_s] \mid v_{s,k} = 1\}$
$\{[\![\mathbf{e}_t]\!]^{\mathcal{C}}_{\mathsf{B}}\}_{t=1}^{h} \leftarrow \mathsf{PrivateExpand}\left(\{[\![j_{k^\star,t}]\!]^{\mathcal{C}}_{\mathsf{B}}\}_{t=1}^{h}\right)$
$[\![\mathbf{P}_s]\!]^{\mathcal{C}}_{\mathsf{B}} \leftarrow \bigoplus_{t=1}^{h/2} [\![\mathbf{e}_t]\!]^{\mathcal{C}}_{\mathsf{B}}, \quad [\![\mathbf{M}_s]\!]^{\mathcal{C}}_{\mathsf{B}} \leftarrow \bigoplus_{t=h/2+1}^{h} [\![\mathbf{e}_t]\!]^{\mathcal{C}}_{\mathsf{B}}$
return $[\![\mathbf{P}_s]\!]^{\mathcal{C}}_{\mathsf{B}}, [\![\mathbf{M}_s]\!]^{\mathcal{C}}_{\mathsf{B}}$

The servers then open the validity bits $\{v_{s,k}\}_{k=1}^{B_s}$, select the first collision-free batch, and retain its Boolean-shared indices $\{[\![j_t]\!]^{\mathcal{C}}_{\mathsf{B}}\}_{t=1}^{h}$. Applying the Private Expansion procedure described in Appendix C.2 yields h Boolean-shared N-dimensional one-hot vectors $\{[\![\mathbf{e}_t]\!]^{\mathcal{C}}_{\mathsf{B}}\}_{t=1}^{h}$. Following the sampling order, the first $h/2$ vectors are aggregated into the positive indicator vector $[\![\mathbf{P}_s]\!]^{\mathcal{C}}_{\mathsf{B}} = \bigoplus_{t=1}^{h/2} [\![\mathbf{e}_t]\!]^{\mathcal{C}}_{\mathsf{B}}$, while the remaining $h/2$ vectors are aggregated into the negative indicator vector $[\![\mathbf{M}_s]\!]^{\mathcal{C}}_{\mathsf{B}} = \bigoplus_{t=h/2+1}^{h} [\![\mathbf{e}_t]\!]^{\mathcal{C}}_{\mathsf{B}}$.

C.4 OT-Based Preprocessing.

For each correlation, C_0 samples $r^{(0)}_{\mathsf{B}} \xleftarrow{\$} \{0,1\}$ and $r^{(0)}_{\ell} \xleftarrow{\$} \mathbb{Z}_{q_\ell}$, while C_1 samples $r^{(1)}_{\mathsf{B}} \xleftarrow{\$} \{0,1\}$. For $b \in \{0,1\}$, C_0 forms $m_{b,\ell} = (r^{(0)}_{\mathsf{B}} \oplus b) - r^{(0)}_{\ell} \pmod{q_\ell}$. A vector 1-out-of-2 OT with the common choice $r^{(1)}_{\mathsf{B}}$ lets C_1 set $r^{(1)}_{\ell} = m_{r^{(1)}_{\mathsf{B}},\ell}$ for all ℓ. We instantiate these OTs in batch using SilentOT.

C.5 Detail of RNS conversion

The Boolean-to-RNS conversion procedure is shown in the Algorithm 4, where $[\![z]\!]^{\mathcal{C}}_{\mathsf{B}}$ denotes the Boolean share to be converted. The two auxiliary servers first locally compute

$$[\![\delta]\!]^{\mathcal{C}}_{\mathsf{B}} \leftarrow [\![z]\!]^{\mathcal{C}}_{\mathsf{B}} \oplus [\![r]\!]^{\mathcal{C}}_{\mathsf{B}},$$

and open δ in one interaction round. Since the Boolean shares to be converted are mutually independent and each conversion uses fresh RNS-consistent correlated bits, all such openings can be batched into the same interaction round.

Once δ is opened, each server $c \in \{0,1\}$ locally computes, for every RNS limb q_ℓ,

$$z^{(c)}_{\ell} \leftarrow (1-2\delta) r^{(c)}_{\ell} + \mathbf{1}[c=0]\delta \pmod{q_\ell}.$$

The resulting $\{z^{(c)}_{\ell}\}_{c\in\{0,1\}}$ form arithmetic shares of z modulo q_ℓ, thereby completing the Boolean-to-RNS conversion without further interaction.

C.6 Detail of Threshold CKKS Key Material Generation

C.6.1 Public Key Generation

Algorithm 5 presents the one-round public-key generation protocol used in our construction. Here, $s^{(i)}_d$ denotes the additive secret-key share held by auxiliary server C_i. The two auxiliary servers

Algorithm 4 RNS-Consistent Boolean-to-RNS Conversion

Input: Boolean-shared value $[\![z]\!]^{\mathcal{C}}_{\mathsf{B}}$ and RNS-Consistent correlated bits $([\![r]\!]^{\mathcal{C}}_{\mathsf{B}}, \{[\![r]\!]^{\mathcal{C}}_{q_\ell}\}_{\ell=0}^{L_{\mathcal{B}}-1})$
Output: $[\![z]\!]^{\mathcal{C}}_{\mathcal{B}}$
1: Compute $[\![\delta]\!]^{\mathcal{C}}_{\mathsf{B}} \leftarrow [\![z]\!]^{\mathcal{C}}_{\mathsf{B}} \oplus [\![r]\!]^{\mathcal{C}}_{\mathsf{B}}$
2: Open $\delta \leftarrow \mathsf{Open}_{\mathsf{B}}([\![\delta]\!]^{\mathcal{C}}_{\mathsf{B}})$
3: For all $(\ell, c) \in \{0, \ldots, L_{\mathcal{B}} - 1\} \times \{0, 1\}$, set
$z_\ell^{(c)} \leftarrow (1 - 2\delta) r_\ell^{(c)} + \mathbf{1}[c = 0]\delta \pmod{q_\ell}$ ▷ local and parallel
4: Set $[\![z]\!]^{\mathcal{C}}_{q_\ell} \leftarrow (z_\ell^{(0)}, z_\ell^{(1)})$ for all ℓ
5: **return** $[\![z]\!]^{\mathcal{C}}_{\mathcal{B}} = \{[\![z]\!]^{\mathcal{C}}_{q_\ell}\}_{\ell=0}^{L_{\mathcal{B}}-1}$

Algorithm 5 Distributed Public-Key Generation

Input: Additive secret-key shares $\{s_d^{(c)}\}_{c\in\{0,1\}}$ such that $s_d = s_d^{(0)} + s_d^{(1)}$.
Output: Public key $pk = (a, b)$.
1: The servers derive a common public $a \in R_Q$ from public randomness.
2: **for** $c \in \{0, 1\}$ **in parallel do**
3: C_c samples $e^{(c)} \leftarrow \chi_e$ and computes $b^{(c)} = -as_d^{(c)} + e^{(c)} \pmod{Q}$.
4: **end for**
5: C_0 and C_1 exchange $b^{(0)}$ and $b^{(1)}$ in one round.
6: Compute $b = b^{(0)} + b^{(1)} \pmod{Q}$ and output $pk = (a, b)$.

locally compute their public-key contributions from these shares and combine them through a single round of interaction to obtain the public key pk.

C.6.2 Relinearization Key Generation

As shown in Algorithm 6, we adapt the two-round multi-party relinearization-key generation protocol of Bossuat et al. (2022) to CKKS. The protocol takes the additive shares $s_d^{(0)}$ and $s_d^{(1)}$ held by the two auxiliary servers and outputs the relinearization key $\mathsf{evk}_{\mathrm{rlin}}$ for the collective secret s_d.

C.6.3 Key-switching Key Generation

We provide a 1-round protocol in Algorithm 7 to generate key-switching keys for two secret keys s_x, s_y shared between two auxiliary servers. The cross-secret switching keys are generated in one round because the CKKS switching-key relation is affine-linear in the source and target secrets, allowing each auxiliary server to locally form a contribution from its secret shares before aggregation. As for rotation keys $\mathsf{evk}_{\mathrm{rot}}$, after a Galois automorphism (Rotation), key switching restores the ciphertext to the original secret domain. The dense–sparse switching keys $\{\mathcal{K}^{qp}_{d\to s}, \mathcal{K}^{QP}_{s\to d}\}$ use the same construction. Notably, the dense-to-sparse key $\mathcal{K}^{qp}_{d\to s}$ is generated only over qp, since this switch occurs at the lowest ciphertext level before $\mathsf{ModRaise}$, while the sparse-to-dense key $\mathcal{K}^{QP}_{s\to d}$ is generated over the full QP modulus.

D Detail of Online Multi-User Inference

Multi-User Input Packing. Following MOAI's interleaved batching Zhang et al. (2026), we pack independently owned inputs encrypted under the collective public key. For an active user set $\mathcal{U} \subseteq [M]$, each user $m \in \mathcal{U}$ is assigned a disjoint slot region Ω_m, such that $\Omega_m \cap \Omega_{m'} = \emptyset, m \neq m'$. Let Place_m align ct_m with Ω_m and zero the remaining user regions. The server constructs

$$ct_{\mathcal{X}} = \mathsf{Pack}_{\mathrm{MU}}\big(\{ct_m\}_{m\in\mathcal{U}}\big) = \sum_{m\in\mathcal{U}} \mathsf{Place}_m(ct_m). \tag{5}$$

Since all inputs share the collective key, the aligned ciphertexts are added directly while preserving the interleaved layout required by the MOAI operators. Figure 3 illustrates the packing of three users under a 256-group layout.

Algorithm 6 Distributed Relinearization-Key Generation

Input: Additive shares $s_d^{(0)}, s_d^{(1)}$ such that $s_d = s_d^{(0)} + s_d^{(1)}$; public CRS vector $\mathbf{a} \in R_{QP}^{\ell}$; public gadget vector $\mathbf{w} \in R_{QP}^{\ell}$.
Output: Relinearization key $\mathsf{evk}_{\mathrm{rlin}}$.

Round 1
for $c \in \{0,1\}$ **in parallel do**
  C_c samples an ternary secret u_c and errors $\mathbf{e}_{0,c}, \mathbf{e}_{1,c}$.
  C_c locally computes $\mathbf{h}_{0,c} = -u_c\mathbf{a} + s_d^{(c)}\mathbf{w} + \mathbf{e}_{0,c}$; $\mathbf{h}_{1,c} = s_d^{(c)}\mathbf{a} + \mathbf{e}_{1,c}$.
end for
The servers exchange and aggregate $\mathbf{h}_0 = \mathbf{h}_{0,0} + \mathbf{h}_{0,1}$; $\mathbf{h}_1 = \mathbf{h}_{1,0} + \mathbf{h}_{1,1}$.
Round 2
for $c \in \{0,1\}$ **in parallel do**
  C_c samples errors $\mathbf{e}_{2,c}, \mathbf{e}_{3,c}$.
  C_c locally computes $\mathbf{h}'_{0,c} = s_d^{(c)}\mathbf{h}_0 + \mathbf{e}_{2,c}$; $\mathbf{h}'_{1,c} = (u_c - s_d^{(c)})\mathbf{h}_1 + \mathbf{e}_{3,c}$.
end for
The servers exchange and aggregate $\mathbf{h}'_0 = \mathbf{h}'_{0,0} + \mathbf{h}'_{0,1}$; $\mathbf{h}'_1 = \mathbf{h}'_{1,0} + \mathbf{h}'_{1,1}$.
return $\mathsf{evk}_{\mathrm{rlin}} = (\mathbf{h}'_0 + \mathbf{h}'_1,\ \mathbf{h}_1)$.

Algorithm 7 Distributed Cross-Secret Switching-Key Generation

Input: Additive shares $\{s_x^{(c)}, s_y^{(c)}\}_{c\in\{0,1\}}$; public CRS vector $\mathbf{a}$ and gadget vector $\mathbf{w}$.
Output: Switching key $\mathsf{K}_{x\to y}$.
for $c \in \{0,1\}$ **in parallel do**
  C_c samples error $\mathbf{e}^{(c)}$ and locally computes $\mathbf{b}^{(c)} = -\mathbf{a}s_y^{(c)} + \mathbf{w}s_x^{(c)} + \mathbf{e}^{(c)}$.
end for
The servers exchange and aggregate $\mathbf{b} = \mathbf{b}^{(0)} + \mathbf{b}^{(1)}$ in one round.
return $\mathsf{K}_{x\to y} = (\mathbf{b}, \mathbf{a})$.

Homomorphic Evaluation. The inference server $\mathcal{S}$ evaluates the packed ciphertext using the packing-aware transformer operators of MOAI Zhang et al. (2026):

$$ct_{\mathcal{Y}} = \mathsf{Eval}_{F_\Theta}\left(ct_{\mathcal{X}}; \mathcal{K}_{\mathrm{pub}}\right). \tag{6}$$

Our framework extends these operators to multi-user ciphertexts under the collective threshold-CKKS infrastructure.

Inference normally remains in the dense-secret domain. During bootstrapping, SSE switches from s_d to s_s using $K_{d\to s}^{qp}$ before $\mathsf{ModRaise}$, performs $\mathsf{ModRaise}$ under s_s, and then switches back to s_d using $K_{s\to d}^{QP}$.

Output Recovery. After inference, the server sends $ct_{\mathcal{Y}}$ and the public slot assignment $\{\Omega_m\}_{m\in\mathcal{U}}$ to the parties. For each active user $m \in \mathcal{U}$, every party derives

$$ct_{y_m} = \mathsf{Extract}_m\left(ct_{\mathcal{Y}}; \Omega_m\right) = (a_{y_m}, b_{y_m}), \tag{7}$$

which applies plaintext slot masking by multiplying the ciphertext with a binary plaintext mask that is 1 at the pre-assigned positions and 0 elsewhere. Ciphertext ct_{y_m} remains encrypted under the collective dense secret s_d. The parties then apply the threshold-decryption protocol of Colin de Verdière et al. (2026) to output each y_m.

Under the M-of-M access structure, every party P_i use their dense secret share $s_d^{(i)}$ to generate a masked partial-decryption share

$$p_{y_m,i} = a_{y_m}s_d^{(i)} + e_{y_m,i}^{\mathrm{smdg}} + r_i^{ct_{y_m}} \pmod{q_\ell}. \tag{8}$$

Here, $e_{y_m,i}^{\mathrm{smdg}} \leftarrow D_{\mathrm{smdg}}$ is fresh, independent smudging noise Boneh et al. (2018), parameterized to statistically hide the bounded ciphertext error even with only one honest decryptor, while preserving the required output precision. Following Colin de Verdière et al. (2026), each pair of parties P_i

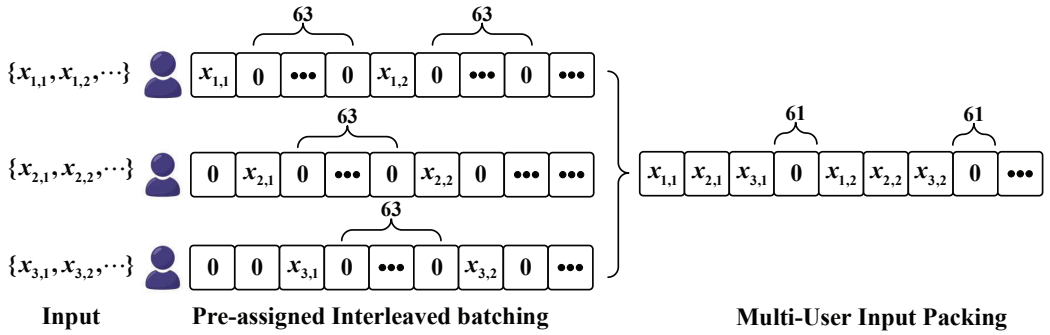

Figure 3: Example of multi-User input packing.

and P_j holds two independent PRF keys $k_{i,j}$ and $k_{j,i}$ established during setup. Using a secure PRF function F with outputs in R_q, party P_i derives

$$r_i^{ct_{y_m}} = \sum_{j \in [M] \setminus \{i\}} \big(F_{k_{i,j}}([M], ct_{y_m}) - F_{k_{j,i}}([M], ct_{y_m})\big). \tag{9}$$

These masks bind the shares to the same ciphertext and the full decryptor set $[M]$, and satisfy $\sum_{i=1}^{M} r_i^{ct_{y_m}} = 0 \pmod{q_\ell}$. Each P_i with $i \neq m$ sends its share directly to P_m over an authenticated confidential channel, while P_m retains its own share locally. Party P_m computes

$$z_{y_m} = b_{y_m} + \sum_{i=1}^{M} p_{y_m,i} = b_{y_m} + a_{y_m} s_d + \sum_{i=1}^{M} e_{y_m,i}^{\mathrm{smdg}} \pmod{q_\ell}, \tag{10}$$

and decodes z_{y_m} to recover the approximate output y_m. Output recovery is therefore determined by the public slot assignment without a separate recipient-selected decryption request. The auxiliary servers C_0 and C_1 remain offline.

E Security Analysis

Based on the static semi-honest model and channel assumptions of Section B, we simulate each permitted adversary's joint view and outputs, first for Setup and Online separately, and then for their state-consistent composition.

E.1 Security Definitions

Let λ be the security parameter. We consider static semi-honest adversaries and fix $\mathcal{I} \subsetneq [M]$, with $\mathcal{H} = [M] \setminus \mathcal{I}$ and active set $\mathcal{U} \subseteq [M]$. Write

$$\mathcal{A}_{\mathrm{set}} := \mathcal{A}^{\mathrm{set}}_{\mathcal{I},b}, \qquad \mathcal{A}_{\mathrm{on}} := \mathcal{A}^{\mathrm{on}}_{\mathcal{I},\mathcal{S}}.$$

The setup adversary corrupts the parties in $\mathcal{I}$ and at most one auxiliary server $\mathcal{C}_b$, with $b \in \{\perp, 0, 1\}$. $b = \perp$ denotes two honest auxiliary servers. The online adversary corrupts the same parties and $\mathcal{S}$. Under the stated phase-separated model, the end-to-end view is $(V^{\mathrm{set}}_{\mathcal{I},\perp}, V^{\mathrm{on}}_{\mathcal{I},\mathcal{S}})$. The model parameters Θ are private input to $\mathcal{S}$ and become available to the adversary through its corruption. We use an M-of-M access structure and authenticated confidential channels.

For $N = 2^n$ and positive even $h \leq N$, define

$$\chi_d = \mathsf{Unif}(\{-1, 0, 1\}^N), \qquad \chi_s = \mathsf{Unif}(\mathcal{S}_{N,h}),$$
$$\mathcal{S}_{N,h} = \{s \in \{-1, 0, 1\}^N : |\{u : s_u = 1\}| = |\{u : s_u = -1\}| = h/2\}.$$

Let ρ_m be P_m's labeled sampling contribution, $s_j = s_d^{(j)}$, and $s_d = \sum_j s_j$. The continuation state is

$$\mathsf{st}_j = (s_j, \{K_{j,k}, K_{k,j}\}_{k \neq j}, \mathsf{cache}_j), \quad \Sigma = (\mathcal{K}_{\mathrm{pub}}, \{\mathsf{st}_j\}_{j=1}^{M}), \quad \mathcal{L}_{\mathrm{setup}} = (\mathbf{v}_d, \mathbf{v}_s, \mathsf{status}). \tag{11}$$

The permitted setup leakage records dense-candidate validity, sparse-batch validity, and success or abort. It is released in the protocol's prescribed order. Let $Z_{\mathcal{I}}$ contain the public configuration and the corrupted parties' retained setup states and transcripts. All claims concern the prescribed joint setup distribution and admissible polynomial-size inputs.

Definition 1 (Setup functionality). *$\mathcal{F}_{\mathrm{Setup}}$ privately receives $\{\rho_m\}$, XORs contributions by label, and applies the prescribed dense rejection sampler and distinct-support sparse sampler. It releases $\mathcal{L}_{\mathrm{setup}}$. On success, it generates, for the same s_d, s_s,*

$$\mathcal{K}_{\mathrm{pub}} = (pk, \mathsf{evk}, K^{\mathrm{low}}_{d\to s}, K^{\mathrm{high}}_{s\to d})$$

according to the joint key-generation reference distribution, and samples uniform RNS-consistent additive shares satisfying $\sum_j s_j = s_d$. It independently generates each ordered-pair key $K_{j,k}$, privately provides it to its two participant endpoints, initializes empty caches, publishes $\mathcal{K}_{\mathrm{pub}}$, and privately shares st_j. The collective secrets remain internal. On sampling abort, it returns $\Sigma = \perp$ with the prescribed visible prefix.

Definition 2 (Online functionality). *After successful setup, $\mathcal{F}^{\mathrm{local}}_{\mathrm{Online}}$ privately receives (sid, Θ) from $\mathcal{S}$ and (sid, m, x_m) from each active party. It computes*

$$\begin{aligned}
\mathcal{X}^{\mathrm{ref}} &= \sum_{m\in\mathcal{U}} \mathsf{Place}^{\mathrm{pt}}_m(x_m),\\
\mathsf{RefOut}_i(\mathbf{x};\Theta) &= \mathsf{Extract}^{\mathrm{pt}}_i\big(F^{\mathrm{ref}}_\Theta(\mathcal{X}^{\mathrm{ref}});\Omega_i\big),\\
v_i &= \mathsf{Encode}_{\Delta_i}\big(\mathsf{RefOut}_i(\mathbf{x};\Theta)\big),\\
\eta_{i,j} &= \mathsf{Samp}_i(\gamma_{i,j}) \sim D_i,\\
w_i &= v_i + \sum_j \eta_{i,j} \in R_i,
\end{aligned} \tag{12}$$

where $R_i = \mathbb{Z}_{Q_i}[X]/(X^N+1)$. The reference operators use the plaintext semantics of the prescribed placement, inference, and extraction, including their layout and numerical approximations. Extraction specifies the entire output encoding. User isolation requires RefOut_i to depend only on (x_i, Θ). The local sampling randomness $\gamma_{i,j}$ is independent across fresh instances and parties. The implemented smudging distribution D_i satisfies

$$\sup_{e\in\mathcal{E}_i} \Delta_{\mathrm{TV}}(D_i, D_i - e) \le \delta_i, \tag{13}$$

for the prescribed pre-smudging error set $\mathcal{E}_i \subseteq R_i$. The functionality privately delivers w_i to P_i, publishes the declared execution metadata μ_{online}, and releases through the corruption interface

$$\Lambda_{\mathcal{I}} = \{(i, j, \gamma_{i,j}, \eta_{i,j}) : i \in \mathcal{U},\ j \in \mathcal{I}\}, \qquad \mathcal{L}^{\mathcal{I}}_{\mathrm{online}} = (\mu_{\mathrm{online}}, \Lambda_{\mathcal{I}}). \tag{14}$$

The public metadata includes the declared session, active set, layout, and input/output shapes. $\Lambda_{\mathcal{I}}$ corresponds to corrupted parties' local sampling records and uses the same samples as $\mathbf{w}$, including contributions to honest recipients. Outputs and records are cached per instance. The authorized output is the raw w_i, followed by deterministic decoding. Security is relative to this explicit local-noise-aware interface.

Definition 3 (Simulation security). *For $t \in \{\mathrm{set}, \mathrm{on}\}$ and every permitted PPT adversary $\mathcal{A}_t$, there must exist a PPT simulator $\mathsf{S}_t^{\mathcal{A}_t}$ such that every admissible input family and PPT distinguisher $\mathcal{D}$ satisfy*

$$\begin{gathered}
\mathsf{REAL}_{t,\mathcal{A}_t} := (\mathsf{View}^{\Pi_t}_{\mathcal{A}_t}, O^{\mathrm{real}}_t), \qquad \mathsf{SIM}_{t,\mathcal{A}_t} := (\mathsf{S}_t^{\mathcal{A}_t}(\Phi_t^{\mathcal{A}_t}), O^{\mathrm{ideal}}_t),\\
\mathrm{Adv}_{\mathcal{D}}\big(\mathsf{REAL}_{t,\mathcal{A}_t}, \mathsf{SIM}_{t,\mathcal{A}_t}\big) \le \varepsilon_t(\lambda),
\end{gathered} \tag{15}$$

where $\mathrm{Adv}_{\mathcal{D}}(X,Y) = |\Pr[\mathcal{D}(X)=1] - \Pr[\mathcal{D}(Y)=1]|$. The authorized interfaces are

$$\begin{gathered}
\Phi^{\mathcal{A}_{\mathrm{set}}}_{\mathrm{set}} = (\mathsf{pp}, \rho_{\mathcal{I}}, \mathcal{K}_{\mathrm{pub}}, \mathcal{L}_{\mathrm{setup}}, \mathsf{st}_{\mathcal{I}}), \quad \Phi^{\mathcal{A}_{\mathrm{on}}}_{\mathrm{on}} = (Z_{\mathcal{I}}, \Theta, \mathbf{x}_{\mathcal{I}}, \mathbf{w}_{\mathcal{I}}, \mathcal{L}^{\mathcal{I}}_{\mathrm{online}}),\\
O_{\mathrm{set}} = \Sigma, \quad O_{\mathrm{on}} = \mathbf{w}.
\end{gathered} \tag{16}$$

Subscripts $\mathcal{I}$ on inputs and outputs select $\mathcal{I}\cap\mathcal{U}$. Views include corrupted inputs, local randomness, persistent states, and observed messages. Ideal interfaces and outputs come from one invocation. The joint comparison with Σ enforces setup output-state consistency. The online simulator receives corrupted recipients' outputs while the experiment includes all outputs.

Definition 4 (System functionality). *$\mathcal{F}_{\mathrm{sys}}^{\mathrm{local}} = (\mathcal{F}_{\mathrm{Setup}}; \mathcal{F}_{\mathrm{Online}}^{\mathrm{local}})$ executes sequentially on the same installed state and stops on setup abort. For a permitted PPT system adversary $\mathcal{A}_{\mathrm{sys}}$, define*

$$V_{\mathrm{sys}}^{\mathcal{A}_{\mathrm{sys}}} = (V_{\mathcal{I},\perp}^{\mathrm{set}}, V_{\mathcal{I},\mathcal{S}}^{\mathrm{on}}), \quad \Phi_{\mathrm{sys}}^{\mathcal{A}_{\mathrm{sys}}} = (\Phi_{\mathrm{set}}^{\mathcal{A}_{\mathcal{I},\perp}^{\mathrm{set}}}, \Theta, \mathbf{x}_{\mathcal{I}}, \mathbf{w}_{\mathcal{I}}, \mathcal{L}_{\mathrm{online}}^{\mathcal{I}}), \quad O_{\mathrm{sys}} = (\mathcal{K}_{\mathrm{pub}}, \mathcal{L}_{\mathrm{setup}}, \mathbf{w}). \tag{17}$$

The system experiments follow equation 15 with these interfaces. Aborted executions use $\perp$ for subsequent outputs.

Underlying guarantees. We distinguish the CKKS error $\epsilon_i \in R_i$ from the security loss ε_t. Let $\mathcal{G}_t$ be a stage's ideal underlying interface and ρ_t its implementation. The bounds $\varepsilon_{\mathsf{B},d}$ and $\varepsilon_{\mathsf{B},s}$ are obtained by instantiating the Boolean MPC circuit-simulation theorem for the dense circuit $\mathcal{C}_d$ and sparse circuit $\mathcal{C}_s$ with the adopted OT or secure-AND backend. $\varepsilon_{\mathrm{Corr}}$ is the correlated-bit backend's joint simulation bound for Boolean/RNS preprocessing and its state-consistent consumption. $\varepsilon_{\mathrm{Key}}$ denotes the joint simulation bound obtained from the adopted key-generation protocols of MHE Mouchet et al. (2021) and their stated CKKS/SSE instantiations, under the prescribed sharing, parameter, and output-consistency conditions. $\varepsilon_{\mathrm{HE}}$ and $\varepsilon_{\mathrm{PRF}}^{\mathrm{hh}}$ come from exposed-state multi-message IND-CPA and honest-endpoint multi-key PRF security. Each bound covers the complete component replacement under its applicable output-consistent composition guarantee.

E.2 Setup Security

Fix an arbitrary permitted PPT adversary $\mathcal{A} = \mathcal{A}_{\mathcal{I},b}^{\mathrm{set}}$. Each $\mathcal{F}_t : X_t \mapsto (L_t, Y_t)$, for $t \in \{\mathrm{Rand}, d, s, R, K, H\}$, is an internal interface of $\mathcal{F}_{\mathrm{Setup}}$. Define

$$\Psi_t^{\mathcal{A}} := (Z_t^{\mathcal{A}}, X_t^{\mathcal{A}}, L_t, Y_t^{\mathcal{A}}), \quad \widehat{V}_t^{\mathcal{A}} \leftarrow \mathsf{S}_t^{\mathcal{A}}(\Psi_t^{\mathcal{A}}), \quad \widehat{Z}_{t^+}^{\mathcal{A}} := \mathsf{Update}_t(Z_t^{\mathcal{A}}, \widehat{V}_t^{\mathcal{A}}, Y_t^{\mathcal{A}}). \tag{18}$$

Here $Z_t^{\mathcal{A}}$ is the preceding adversarial state, superscript $\mathcal{A}$ denotes the corrupted projection, and t^+ denotes the next stage. The update follows the real local state transitions. The stage experiments compare

$$\mathsf{REAL}_{t,\mathcal{A}} := (Z_t^{\mathcal{A}}, V_t^{\mathcal{A}}, Y_t^{\mathrm{real}}), \qquad \mathsf{SIM}_{t,\mathcal{A}} := (Z_t^{\mathcal{A}}, \mathsf{S}_t^{\mathcal{A}}(\Psi_t^{\mathcal{A}}), Y_t^{\mathrm{ideal}}). \tag{19}$$

Intermediate outputs are connected through the stipulated output-consistent interfaces. The complete setup simulator retains the external input $\Phi_{\mathrm{set}}^{\mathcal{A}}$.

Lemma 1 (Stage reduction). *Suppose ρ_t realizes $\mathcal{G}_t$ with error $\varepsilon_{\mathrm{base},t}$ and $\Pi_t^{\mathcal{G}_t}$ realizes $\mathcal{F}_t$ with error δ_t^{hyb}. Under output-consistent sequential composition, there exists $\mathsf{S}_t^{\mathcal{A}}$ such that*

$$\mathrm{Adv}_{\mathcal{D}}(\mathsf{REAL}_{t,\mathcal{A}}, \mathsf{SIM}_{t,\mathcal{A}}) \le \varepsilon_{\mathrm{base},t} + \delta_t^{\mathrm{hyb}}. \tag{20}$$

Proof. Starting from the real stage, replace ρ_t by $\mathcal{G}_t$ and its backend simulator, then replace the resulting stage by $\mathcal{F}_t$ and its ideal-interface simulator. The respective losses are $\varepsilon_{\mathrm{base},t}$ and δ_t^{hyb}. Composing the simulators on the same state gives $\mathsf{S}_t^{\mathcal{A}}$. The triangle inequality proves the bound, jointly with the preceding context and outputs. □

Lemma 2 (Contribution randomness). *Suppose an honest party supplies independent uniform bits at every label, with independent sharing masks. For the post-sharing, pre-sampling adversarial view Z_0 and any label set A,*

$$\Pr[(u_\alpha)_{\alpha \in A} = z \mid Z_0] = 2^{-|A|}. \tag{21}$$

Contribution sharing perfectly realizes $\mathcal{F}_{\mathrm{Rand}}$, which privately delivers Boolean shares of $u_\alpha = \bigoplus_m u_{m,\alpha}$.

Proof. Simulator. $\mathsf{S}_{\mathrm{Rand}}^{\mathcal{A}}$ reproduces corrupted senders' shares $(q_{j,\alpha}, u_{j,\alpha} \oplus q_{j,\alpha})$ using consistent local randomness. If C_b is corrupted, fix $h_\star \in \mathcal{H}$, sample the other honest incoming shares uniformly, and set

$$\widehat{u}_{h_\star,\alpha}^{(b)} = u_\alpha^{(b)} \oplus \bigoplus_{m \neq h_\star} \widehat{u}_{m,\alpha}^{(b)}. \tag{22}$$

It records the messages and prescribed aggregate share. For $b = \perp$, the simulated view consists of corrupted parties' sending records.

Indistinguishability. Let $u_{h,\alpha}$ be an honest party's uniform contribution and $q_{h,\alpha}$ an independent uniform sharing mask. The shares are $u^{(0)}_{h,\alpha} = q_{h,\alpha}, u^{(1)}_{h,\alpha} = u_{h,\alpha} \oplus q_{h,\alpha}$. An honest contribution and either share satisfy $\Pr[u_{h,\alpha} = a, u^{(b)}_{h,\alpha} = c] = 1/4$. Label independence and XOR invariance yield equation 21. Equation equation 22 samples the exact conditional law given the aggregate. Hence

$$\mathsf{REAL}_{\mathrm{Rand},\mathcal{A}} \equiv_d \mathsf{SIM}_{\mathrm{Rand},\mathcal{A}}, \qquad \varepsilon_{\mathrm{Rand}} = 0.$$

□

Lemma 3 (Dense-secret sampling)**.** *$\mathcal{F}_d$ reveals $\mathbf{v}_d$ and delivers the prescribed Boolean shares of the selected indicators (P_d, M_d). Under Lemma 2, for every positive-probability successful pattern $\bar{\mathbf{v}}_d$ and $z \in \{-1, 0, 1\}^N$,*

$$\Pr[s_d = z \mid \mathbf{v}_d = \bar{\mathbf{v}}_d, Z_0] = 3^{-N}. \tag{23}$$

The stage has zero sampling bias and simulation error at most $\varepsilon_{\mathsf{B},d}$.

Proof. Simulator. Let $\mathcal{C}_d$ be the Boolean circuit for dense-candidate validity and selection of the first N accepted candidates, and let $\mathsf{Sim}_{\mathsf{B}}$ be the adopted Boolean MPC backend's output-consistent circuit simulator.

For a corrupted C_b, the active input and output projections are

$$X^{\mathcal{A}}_d = \big((a^{(b)}_k, b^{(b)}_k)\big)_k, \qquad Y^{\mathcal{A}}_d = (P^{(b)}_d, M^{(b)}_d),$$

where superscript (b) denotes the Boolean share held by C_b, k ranges over the candidate bit pairs, and P_d, M_d are the selected positive and negative indicator vectors. The public output $L_d = \mathbf{v}_d$ records candidate validity.

The simulator derives the increasing list π of accepted indices and the status from $\mathbf{v}_d$, then invokes

$$(\widehat{\tau}^{\mathcal{A}}_d, \widehat{\omega}^{\mathcal{A}}_d) \leftarrow \mathsf{Sim}_{\mathsf{B}}\big(\mathcal{C}_d; Z^{\mathcal{A}}_d, X^{\mathcal{A}}_d, \mathbf{v}_d, Y^{\mathcal{A}}_d\big). \tag{24}$$

Here $\widehat{\tau}^{\mathcal{A}}_d$ contains the simulated validity computation and opening messages. $\widehat{\omega}^{\mathcal{A}}_d$ records the corresponding local randomness, wire shares, and selections. The simulated opening shares satisfy

$$\widehat{v}^{(1-b)}_{d,k} = v_{d,k} \oplus \widehat{v}^{(b)}_{d,k}.$$

The simulator records the prescribed output or abort prefix and updates the existing state. For $b = \perp$, it takes the public and corrupted-party projection. On abort, $Y^{\mathcal{A}}_d = \perp$.

Indistinguishability. The circuit computes

$$[\![v_{d,k}]\!] = \neg\mathsf{AND}_{\mathsf{B}}([\![a_k]\!], [\![b_k]\!]), \tag{25}$$

so each uniform candidate is accepted with probability $3/4$ and each accepted value has conditional probability $1/3$. Conditioning on the complete validity pattern preserves candidate independence and fixes π, giving equation 23. In the ideal underlying model, the circuit simulator preserves the joint input-share, opening, and output law. Selection is deterministic post-processing. Thus $\delta^{\mathrm{hyb}}_d = 0$, and Lemma 1 gives

$$\mathrm{Adv}_{\mathcal{D}}(\mathsf{REAL}_{d,\mathcal{A}}, \mathsf{SIM}_{d,\mathcal{A}}) \le \varepsilon_{\mathsf{B},d}.$$

□

Lemma 4 (Sparse-secret sampling and expansion)**.** *$\mathcal{F}_s$ reveals $\mathbf{v}_s$ and delivers shares of (P_s, M_s) for the first valid batch. Under the same randomness conditions, every positive-probability successful pattern $\bar{\mathbf{v}}_s$ and $s \in \mathcal{S}_{N,h}$ satisfy*

$$\Pr[s_s = s \mid \mathbf{v}_s = \bar{\mathbf{v}}_s, Z_0] = \frac{((h/2)!)^2}{(N)_h} = \frac{1}{\binom{N}{h}\binom{h}{h/2}}, \tag{26}$$

where $(N)_h = N(N-1)\cdots(N-h+1)$. The stage has zero sampling bias and simulation error at most $\varepsilon_{\mathsf{B},s}$.

Proof. Simulator. Let $\mathcal{C}_s$ be the Boolean circuit for collision testing, selection of the first valid batch, and private one-hot expansion. For a corrupted C_b, its input and output projections are

$$X_s^{\mathcal{A}} = \big(\mathbf{J}_k^{(b)}\big)_k, \qquad Y_s^{\mathcal{A}} = (P_s^{(b)}, M_s^{(b)}),$$

where $\mathbf{J}_k^{(b)} = (J_{k,1}^{(b)}, \ldots, J_{k,h}^{(b)})$ contains its bitwise Boolean shares of the h candidate indices in batch k, and P_s, M_s are the positive and negative support indicators. The public output $L_s = \mathbf{v}_s$ records whether each batch has distinct indices.

The simulator derives $k^\star = \min\{k : v_{s,k} = 1\}$ or the abort status, then invokes the same Boolean MPC circuit simulator:

$$(\widehat{\tau}_s^{\mathcal{A}}, \widehat{\omega}_s^{\mathcal{A}}) \leftarrow \mathsf{Sim}_{\mathsf{B}}\big(\mathcal{C}_s; Z_s^{\mathcal{A}}, X_s^{\mathcal{A}}, \mathbf{v}_s, Y_s^{\mathcal{A}}\big). \tag{27}$$

Here $\widehat{\tau}_s^{\mathcal{A}}$ contains the simulated validation, opening, and expansion messages. $\widehat{\omega}_s^{\mathcal{A}}$ records local randomness and the intermediate equality, index, and one-hot shares. On success, the simulator records the local XORs of the first and last $h/2$ one-hot shares, consistent with $Y_s^{\mathcal{A}}$. On abort, the prescribed output is $Y_s^{\mathcal{A}} = \perp$ and the simulator returns the corresponding prefix. It then updates the shared state. For $b = \perp$, it takes the public and corrupted-party projection.

Indistinguishability. Each n-bit index is uniform on $[N]$. Conditioning a batch on distinctness makes its ordered support uniform over $(N)_h$ tuples. Batch independence preserves this law after conditioning on the full validity pattern and selecting $k^\star$. Every balanced weight-h secret has exactly $((h/2)!)^2$ ordered preimages, proving equation 26. Distinctness also gives $\mathrm{HW}(P_s) = \mathrm{HW}(M_s) = h/2$ and $P_s \odot M_s = 0$. These identities establish the circuit/function correspondence. Ideal-interface circuit simulation has $\delta_s^{\mathrm{hyb}} = 0$. Lemma 1 therefore yields

$$\mathrm{Adv}_{\mathcal{D}}(\mathsf{REAL}_{s,\mathcal{A}}, \mathsf{SIM}_{s,\mathcal{A}}) \le \varepsilon_{\mathsf{B},s}.$$

□

Lemma 5 (RNS-consistent conversion)**.** *$\mathcal{F}_R$ converts Boolean shares of a bit into the prescribed RNS additive shares. Treating correlated-bit generation and consumption as one linked interface, conversion is perfectly simulatable with ideal preprocessing and has error at most $\varepsilon_{\mathrm{Corr}}$ with the adopted backend.*

Proof. Simulator. For corrupted C_b, $\mathsf{S}_R^{\mathcal{A}}$ takes the Boolean input share $z_B^{(b)}$ and prescribed arithmetic outputs $\{z_\ell^{(b)}\}_\ell$, samples $\widehat{d}, \widehat{r}_B^{(b)}$ uniformly, and sets

$$\widehat{r}_\ell^{(b)} = (1 - 2\widehat{d})(z_\ell^{(b)} - \mathbf{1}[b = 0]\widehat{d}) \pmod{q_\ell}, \quad \widehat{d}^{(b)} = z_B^{(b)} \oplus \widehat{r}_B^{(b)}, \quad \widehat{d}^{(1-b)} = \widehat{d} \oplus \widehat{d}^{(b)}. \tag{28}$$

It invokes the output-consistent preprocessing simulator with these prescribed shares, records the opening and arithmetic outputs, and reuses each preprocessing identifier across generation and consumption. It then records $s_x^{(b)} = P_x^{(b)} - M_x^{(b)}$ for $x \in \{d, s\}$.

Indistinguishability. Real conversion satisfies

$$z_\ell^{(c)} = (1 - 2d) r_\ell^{(c)} + \mathbf{1}[c = 0] d \pmod{q_\ell}, \qquad d = z \oplus r, \tag{29}$$

so $\sum_c z_\ell^{(c)} = d + (1 - 2d)r = z$ on every limb. Ideal preprocessing shares the same bit r using independent uniform sharing masks. Equation equation 29 is bijective in each arithmetic mask. Conditioned on the outputs, $d, r_B^{(b)}$ remain independent and uniform. Thus equation 28 samples the exact conditional law, and $\delta_R^{\mathrm{hyb}} = 0$. Replacing the linked preprocessing backend gives

$$\mathrm{Adv}_{\mathcal{D}}(\mathsf{REAL}_{R,\mathcal{A}}, \mathsf{SIM}_{R,\mathcal{A}}) \le \varepsilon_{\mathrm{Corr}}.$$

□

Lemma 6 (CKKS/SSE key generation)**.** *The auxiliary servers instantiate the collective key-generation protocols of Mouchet et al. (2021) [Sections 4.3, 4.4, 4.9 and Appendix B.1] using the RNS shares s_d, s_s established by the preceding stages to generates the complete $\mathcal{K}_{\mathrm{pub}}$. The SSE switching keys use the corresponding source–target secret-sharing adaptation. Under the security conditions of these instantiations and the stated output-consistent composition guarantee, the stage realizes $\mathcal{F}_K$ with simulation error at most $\varepsilon_{\mathrm{Key}}$, the combined bound of the adopted protocol simulations.*

Proof. Simulator. Fix the permitted adversary $\mathcal{A} = \mathcal{A}^{\mathrm{set}}_{\mathcal{I},b}$. The simulator receives

$$\Psi^{\mathcal{A}}_K = \left(Z^{\mathcal{A}}_K, X^{\mathcal{A}}_K, \mathcal{K}_{\mathrm{pub}}, Y^{\mathcal{A}}_K\right),$$

where $X^{\mathcal{A}}_K$ contains the corrupted auxiliary server's temporary shares $\sigma_{d,b}$ and $\sigma_{s,b}$ of s_d and s_s, and $Y^{\mathcal{A}}_K$ is its prescribed retained state.

It instantiates the published EncKeyGen and RelinKeyGen simulators and the corresponding rotation-key adaptation, using the same corrupted input shares and prescribed public key outputs. The two rounds of relinearization-key generation are simulated jointly. Corrupted local sampling records are preserved consistently with the backend's private-input convention.

For an SSE switching-key component from s_x to s_y, the prescribed relation is

$$\begin{aligned} B_{\ell,c} &= -A_\ell \sigma_{y,c} + g_\ell \sigma_{x,c} + E_{\ell,c}, \qquad c \in \{0,1\}, \\ B_\ell &= B_{\ell,0} + B_{\ell,1}. \end{aligned} \tag{30}$$

Here A_ℓ is the public random component, g_ℓ is the prescribed gadget and modulus factor, and $E_{\ell,c}$ is the local key-generation error.

For corrupted C_b, the simulator retains its contribution $B_{\ell,b}$ and sets

$$\widehat{B}_{\ell,1-b} = B_\ell - B_{\ell,b}. \tag{31}$$

It applies this rule to both switching directions at their prescribed moduli.

The resulting messages and local records form $(\widehat{\tau}^{\mathcal{A}}_K, \widehat{\omega}^{\mathcal{A}}_K)$. All records use the same secret-share identifiers and the same ideal invocation producing $\mathcal{K}_{\mathrm{pub}}$. The simulator then passes the prescribed retained state to ownership transfer. For $b = \perp$, it returns the corresponding permitted public and corrupted-party projection.

Indistinguishability. The adopted MHE protocol simulations provide the corresponding key-generation transcript bounds. For the affine SSE contributions, equation 31 reproduces the honest contribution exactly once the corrupted local record and public aggregate are fixed.

Combining these simulations in protocol order preserves the shared state and prescribed public outputs. Their output-consistent composition therefore gives

$$\mathrm{Adv}_{\mathcal{D}}\left(\mathsf{REAL}_{K,\mathcal{A}}, \mathsf{SIM}_{K,\mathcal{A}}\right) \le \varepsilon_{\mathrm{Key}}.$$

□

Lemma 7 (Ownership transfer and decryption-state initialization)**.** *$\mathcal{F}_H$ privately delivers uniform owner shares with $\sum_j s_j = s_d$ and independent endpoint keys. Under exact uniform ring sampling and ideal confidential channels, the prescribed implementation is perfectly simulatable.*

Proof. Simulator. Let $\sigma_0 + \sigma_1 = s_d$ and $\mathcal{T}(x) = \{z \in R_Q^M : \sum_j z_j = x\}$. Given corrupted C_b's input σ_b, $\mathsf{S}^{\mathcal{A}}_H$ samples $\boldsymbol{\tau}_b \leftarrow \mathsf{Unif}(\mathcal{T}(\sigma_b))$, records its sending row, and sets

$$\widehat{\tau}_{1-b,j} = s_j - \tau_{b,j}, \qquad j \in \mathcal{I}. \tag{32}$$

For $b = \perp$, it samples $\widehat{\tau}_{0,j}$ uniformly and sets $\widehat{\tau}_{1,j} = s_j - \widehat{\tau}_{0,j}$. It records each corrupted owner's incoming messages and local sum, reproduces initialization records using the prescribed endpoint keys, and installs $\mathsf{st}_{\mathcal{I}}$ with empty caches.

Indistinguishability. For fixed $\boldsymbol{\tau}_b$, translation maps $\mathcal{T}(\sigma_{1-b})$ bijectively onto $\mathcal{T}(s_d)$. Thus the complete owner sharing is uniform and, conditioned on the auxiliary server inputs, independent of $\boldsymbol{\tau}_b$. Equation equation 32 gives the exact conditional messages. Every proper-subset projection has $|R_Q|^{M-|\mathcal{I}|-1}$ completions, establishing the $b = \perp$ case. Independent endpoint keys preserve this joint law. Therefore

$$\mathsf{REAL}_{H,\mathcal{A}} \equiv_d \mathsf{SIM}_{H,\mathcal{A}}, \qquad \varepsilon_H = 0.$$

□

Theorem 1 (Setup security)**.** *Under the preceding stage assumptions and the stipulated output-consistent composition guarantee for the connected shared-state interfaces, every permitted PPT adversary $\mathcal{A}_{\mathrm{set}}$ admits a PPT simulator satisfying Definition 3 for $\mathcal{F}_{\mathrm{Setup}}$, with*

$$\varepsilon_{\mathrm{set}} \le \varepsilon_{\mathsf{B},d} + \varepsilon_{\mathsf{B},s} + \varepsilon_{\mathrm{Corr}} + \varepsilon_{\mathrm{Key}}. \tag{33}$$

Proof. Simulator. $\mathsf{S}^{\mathcal{A}}_{\mathrm{set}}(\Phi^{\mathcal{A}}_{\mathrm{set}})$ connects the six stage simulators using one table of shares, public records, preprocessing identifiers, and continuation states. The assumed output-consistent composition supplies compatible intermediate states jointly with Σ. Records follow the real schedule, including the abort prefix.

Indistinguishability. Let $G^{\mathcal{A}}_t = (V^{(t)}_{\mathcal{I},b}, \Sigma^{(t)})$. $G^{\mathcal{A}}_0$ is real setup. $\{G^{\mathcal{A}}_1, \dots, G^{\mathcal{A}}_6\}$ successively replace contribution sharing, dense sampling, sparse sampling and expansion, RNS conversion, key generation, and ownership/state initialization. Their losses are respectively

$$(0, \varepsilon_{\mathrm{B},d}, \varepsilon_{\mathrm{B},s}, \varepsilon_{\mathrm{Corr}}, \varepsilon_{\mathrm{Key}}, 0).$$

The final hybrid equals $\mathsf{SIM}_{\mathrm{set},\mathcal{A}}$ in distribution, proving equation 33 by the triangle inequality. The matched sampling-abort branch adds zero distinguishing loss. □

E.3 Online Security

Fix an arbitrary permitted PPT adversary $\mathcal{A} = \mathcal{A}^{\mathrm{on}}_{\mathcal{I},\mathcal{S}}$.

Theorem 2 (Online security)**.** *Assume multi-message IND-CPA security of the collective CKKS/SSE scheme given $(Z_{\mathcal{I}}, \Theta)$ and all published evaluation keys, with bound $\varepsilon_{\mathrm{HE}}$. Let the complete-graph endpoint masks have joint replacement bound*

$$\varepsilon_{\mathrm{mask}}(\mathcal{I}) = \begin{cases} \varepsilon^{\mathrm{hh}}_{\mathrm{PRF}}, & |\mathcal{I}| \le M-2, \\ 0, & |\mathcal{I}| = M-1. \end{cases} \tag{34}$$

For the extracted $ct_{y_i} = (a_{y_i}, b_{y_i})$, define

$$\epsilon_i = b_{y_i} + a_{y_i} s_d - v_i, \qquad \mathsf{Bad}_{\mathrm{dec}} := \{\exists i \in \mathcal{U} : \epsilon_i \notin \mathcal{E}_i\}, \qquad \Pr[\mathsf{Bad}_{\mathrm{dec}}] \le \beta_{\mathrm{dec}}. \tag{35}$$

Here β_{dec} bounds the probability that a pre-smudging error exceeds its prescribed range after inference and extraction. A deterministic error bound gives $\beta_{\mathrm{dec}} = 0$. If equation 13 holds and fresh smudging is independent of the pre-smudging state, the protocol realizes Definition 2 against every such $\mathcal{A}$, with

$$\varepsilon_{\mathrm{on}} \le \varepsilon_{\mathrm{HE}} + \varepsilon_{\mathrm{mask}}(\mathcal{I}) + \beta_{\mathrm{dec}} + \sum_{i \in \mathcal{U}} \delta_i. \tag{36}$$

Proof. Simulator. $\mathsf{S}^{\mathcal{A}}_{\mathrm{on}}(\Phi^{\mathcal{A}}_{\mathrm{on}})$ preserves the setup state, encrypts corrupted users' actual inputs and honest users' admissible same-shape dummies, and runs the prescribed packing, inference, and extraction using the corrupted server's Θ. This gives $\widehat{ct}_{y_i} = (\widehat{a}_{y_i}, \widehat{b}_{y_i})$ and consistent records of encryption and server randomness. Using the same $\Lambda_{\mathcal{I}}$ and corrupted endpoint keys, it computes

$$\widehat{p}_{i,j} = \widehat{a}_{y_i} s_j + \eta_{i,j} + \widehat{r}_{i,j}, \qquad \widehat{r}_{i,j} = \sum_{k \ne j} \big(F^{(i)}_{K_{j,k}}(\widehat{t}_i) - F^{(i)}_{K_{k,j}}(\widehat{t}_i)\big), \tag{37}$$

where $j \in \mathcal{I}$. The PRF range is R_i; endpoint keys and sampling randomness are independent. Tags bind the key epoch, session, recipient, ciphertext, ring, scale, and decryptor set. Fresh instances have distinct tags and retransmissions reuse cached values.

Fix $h_0 \in \mathcal{H}$. For each corrupted recipient, sample $\widehat{p}_{i,h} \leftarrow R_i$ uniformly for $h \in \mathcal{H} \setminus \{h_0\}$ and set

$$\widehat{p}_{i,h_0} = w_i - \widehat{b}_{y_i} - \sum_{j \ne h_0} \widehat{p}_{i,j}. \tag{38}$$

For honest recipients, the visible delivery records are those of corrupted senders. Honest messages remain on confidential channels, and recipients include their own contributions locally. All recipients share one simulation state and cache.

Hybrids. Let $H^{\mathcal{A}}_t = (V^{(t)}_{\mathcal{I},\mathcal{S}}, \mathbf{W}^{(t)})$, for $t = 0, 1, 2, 3$, with each view retaining $Z_{\mathcal{I}}$.

$H^{\mathcal{A}}_0$: real execution. This is $\mathsf{REAL}_{\mathrm{on},\mathcal{A}}$, with $W^{(0)}_i = v_i + \epsilon_i + \sum_j \eta_{i,j}$.

$H^{\mathcal{A}}_1$: reference-output hybrid. Keep the real ciphertexts and corrupted local randomness, and replace one honest contribution by $\eta_{i,h_0} - \epsilon_i$. Conditioned on a good pre-smudging state and corrupted

randomness, the shifts are fixed and the honest noises retain their product law. Equations equation 13 and equation 35, followed by data processing, give

$$\Delta_{\mathrm{TV}}(H_0^{\mathcal{A}}, H_1^{\mathcal{A}}) \le \beta_{\mathrm{dec}} + \sum_i \delta_i, \qquad W_i^{(1)} = v_i + \sum_j \eta_{i,j} = w_i.$$

All outputs retain their ideal correlation with the unchanged $\Lambda_{\mathcal{I}}$.

$H_2^{\mathcal{A}}$: output-consistent share hybrid. If $|\mathcal{I}| \le M-2$, replace honest–honest PRFs by consistent random functions, at cost $\varepsilon_{\mathrm{PRF}}^{\mathrm{hh}}$. After fixing non-tree edges, uniform edge differences on an honest spanning tree bijectively parameterize zero-sum vectors over R_i. Consequently, conditional on corrupted records and w_i, honest shares are uniform on

$$\left\{ (q_h)_{h\in\mathcal{H}} : \sum_h q_h = w_i - b_{y_i} - \sum_{j\in\mathcal{I}} p_{i,j} \right\}.$$

Equation equation 38 samples this law exactly.

If $|\mathcal{I}| = M-1$, the adversary can compute $r_{i,h_0} = -\sum_{j\in\mathcal{I}} r_{i,j}$. Set $H_2^{\mathcal{A}} = H_1^{\mathcal{A}}$. The sole honest share already equals

$$a_{y_i} s_{h_0} + \eta_{i,h_0} + r_{i,h_0} - \epsilon_i = w_i - b_{y_i} - \sum_{j\in\mathcal{I}} p_{i,j}.$$

The residual simulation is exact and the PRF replacement loss is zero. Both cases preserve the joint law of honest outputs and corrupted sampling records, giving

$$\mathrm{Adv}_{\mathcal{D}}(H_1^{\mathcal{A}}, H_2^{\mathcal{A}}) \le \varepsilon_{\mathrm{mask}}(\mathcal{I}).$$

$H_3^{\mathcal{A}}$: dummy-input ciphertext hybrid. Replace honest encryptions by admissible same-shape dummy encryptions, recompute the transcript using Θ, and rerun the delivery simulator. Keep $(\mathbf{w}, \Lambda_{\mathcal{I}})$ jointly generated from the actual inputs. An encryption reduction knows its actual/dummy candidate messages and Θ, so it can generate this ideal auxiliary information independently of the challenge bit and simulate delivery. Exposed-state multi-message IND-CPA yields

$$\mathrm{Adv}_{\mathcal{D}}(H_2^{\mathcal{A}}, H_3^{\mathcal{A}}) \le \varepsilon_{\mathrm{HE}}.$$

The final simulator uses exactly $\Phi_{\mathrm{on}}^{\mathcal{A}}$, and $H_3^{\mathcal{A}} \equiv_d \mathsf{SIM}_{\mathrm{on},\mathcal{A}}$. The triangle inequality proves equation 36. The first transition handles approximate-decryption error leakage captured by IND-CPA^D Li & Micciancio (2021). Deterministic decoding preserves indistinguishability and application precision is governed by the corresponding correctness bound. □

E.4 System Security

Theorem 3 (State-consistent system security)**.** *Assume Theorem 1 provides output-state-consistent security and Theorem 2 applies with the complete permitted setup record. Every permitted PPT system adversary admits a PPT simulator realizing Definition 4, with*

$$\begin{aligned} \varepsilon_{\mathrm{sys}} &\le \varepsilon_{\mathrm{set}} + \varepsilon_{\mathrm{on}} \\ &\le \varepsilon_{\mathsf{B},d} + \varepsilon_{\mathsf{B},s} + \varepsilon_{\mathrm{Corr}} + \varepsilon_{\mathrm{Key}} + \varepsilon_{\mathrm{HE}} + \varepsilon_{\mathrm{mask}}(\mathcal{I}) + \beta_{\mathrm{dec}} + \sum_{i\in\mathcal{U}} \delta_i. \end{aligned} \tag{39}$$

Negligible terms give computational indistinguishability of the complete real and simulated system experiments.

Proof. Fix an arbitrary permitted $\mathcal{A} = \mathcal{A}_{\mathrm{sys}}$. The simulator runs the setup simulator, then supplies the resulting adversarial records and the same state to the online simulator. Both phases use the same public keys, corrupted shares, and endpoint keys. On success, the ideal functionality supplies $(\mathbf{w}_{\mathcal{I}}, \mathcal{L}_{\mathrm{online}}^{\mathcal{I}})$. On abort, the simulator returns the prescribed prefix.

Starting from $J_0^{\mathcal{A}} = \mathsf{REAL}_{\mathrm{sys},\mathcal{A}}$, $J_1^{\mathcal{A}}$ replaces setup while retaining its complete continuation state, at cost $\varepsilon_{\mathrm{set}}$. $J_2^{\mathcal{A}}$ then replaces the online execution with the same retained information, at cost $\varepsilon_{\mathrm{on}}$. Since $J_2^{\mathcal{A}} \equiv_d \mathsf{SIM}_{\mathrm{sys},\mathcal{A}}$, state-consistent sequential composition gives equation 39. □

E.5 Noise-Flooding Parameter Selection

OpenFHE Badawi et al. (2022) calibrates the coefficient standard deviation of Gaussian flooding noise as

$$\sigma_i^{\mathrm{OF}} = \sqrt{12\tau}\, 2^{s_F/2} B_i^{\mathrm{cal}}. \tag{40}$$

Here σ_i^{OF} is the calibrated flooding-noise standard deviation for ciphertext i, B_i^{cal} is its pre-flooding ciphertext-error estimate in OpenFHE's normalization, s_F is OpenFHE's statistical-security parameter in bits, and τ is the anticipated number of adversarial decryption queries.

E.6 Sparse-Secret Parameter Security

SSE uses asymmetric switching keys: $\{K_{d\to s}^{qp}, K_{s\to d}^{QP}\}$. The low-level key $K_{d\to s}^{qp}$ contains RLWE samples encrypted under the sparse secret s_s. Its security is therefore evaluated using the ring dimension N, the low key-switching modulus qp, the sparse-secret distribution determined by h, and the adopted error distribution. In contrast, $K_{s\to d}^{QP}$ contains RLWE samples encrypted under the dense secret s_d. It and the remaining high-modulus evaluation material are governed by the dense-secret security bound at QP.

Although ModRaise produces $ct_{s_s}^{Q}$, this ciphertext is a deterministic lift of $ct_{s_s}^{q}$ rather than a freshly sampled high-modulus RLWE instance under s_s. Consequently, the sparse-secret security requirement is confined to the low-modulus qp regime, whereas the full-modulus QP regime remains protected by the dense-secret security bound. Both parameter sets must meet the target security level.

F Detailed Complexity and Failure Analysis

This supplement provides the detailed operation counts, interaction depths, and sampling-failure bounds summarized in Section 4.5. We count interactive rounds between the two auxiliary servers after the parties' randomness contributions have been received. Offline correlated randomness and preprocessing are not included in the online round count.

Let

$$n = \log_2 N, \qquad J_h = \binom{h}{2}, \tag{41}$$

where h denotes the sparse-secret Hamming weight. Let T_d be the number of dense-secret candidates, B_s the number of sparse-support candidate batches, and λ_{stat} the target statistical sampling-failure parameter.

F.1 Dense-Secret Generation

Each dense-secret candidate is represented by two uniformly distributed Boolean-shared bits $(a, b) \in \{0,1\}^2$. The state $(1,1)$ is invalid, and its validity can therefore be determined using one secure Boolean AND:

$$\mathsf{invalid} = a \wedge b. \tag{42}$$

Hence, evaluating T_d candidates requires exactly

$$C_d = T_d \tag{43}$$

secure AND operations.

All candidate-validity tests are independent and can be evaluated in parallel in one Boolean-AND layer. The resulting validity bits are then opened in one batch. Under our Boolean-MPC realization, the online interaction depth is therefore

$$\mathsf{rnd}_d = 2. \tag{44}$$

Once the validity pattern is public, the first N valid candidates can be selected by public indexing, requiring no additional secure Boolean operations.

Let $v_{d,k} \in \{0,1\}$ indicate whether candidate k is valid. Since

$$\Pr[v_{d,k} = 1] = \frac{3}{4}, \tag{45}$$

the number of valid candidates

$$X_d := \sum_{k=1}^{T_d} v_{d,k} \tag{46}$$

satisfies

$$X_d \sim \text{Binomial}\left(T_d, \frac{3}{4}\right). \tag{47}$$

Dense sampling fails only when fewer than N valid candidates are obtained:

$$p_{\text{fail}}^d = \Pr[X_d < N]. \tag{48}$$

Equivalently,

$$p_{\text{fail}}^d = \sum_{j=0}^{N-1} \binom{T_d}{j} \left(\frac{3}{4}\right)^j \left(\frac{1}{4}\right)^{T_d - j}. \tag{49}$$

We therefore choose

$$T_d = \min\left\{T \geq N : \Pr\left[\text{Binomial}\left(T, \frac{3}{4}\right) < N\right] \leq 2^{-(\lambda_{\text{stat}}+1)}\right\}. \tag{50}$$

F.2 Sparse-Support Validation

Each sparse-secret candidate batch contains h independently sampled n-bit indices. A batch is valid if all h indices are distinct. There are

$$J_h = \binom{h}{2} \tag{51}$$

pairs that must be compared.

For two indices

$$\mathbf{u} = (u_0, \ldots, u_{n-1}), \qquad \mathbf{v} = (v_0, \ldots, v_{n-1}), \tag{52}$$

the bit-wise equality indicators are obtained locally as

$$e_i = 1 \oplus u_i \oplus v_i. \tag{53}$$

The equality bit is then

$$\mathsf{eq}(\mathbf{u}, \mathbf{v}) = \bigwedge_{i=0}^{n-1} e_i. \tag{54}$$

A binary reduction tree requires

$$n - 1 \tag{55}$$

secure AND operations and has interaction depth

$$\lceil \log_2 n \rceil. \tag{56}$$

For one candidate batch, the J_h pairwise equality results are combined into a collision indicator. Using

$$x \vee y = x \oplus y \oplus (x \wedge y), \tag{57}$$

a binary OR tree requires

$$J_h - 1 \tag{58}$$

additional secure AND operations and interaction depth

$$\lceil \log_2 J_h \rceil. \tag{59}$$

Hence, validating one candidate batch requires

$$J_h(n-1) + (J_h - 1) = J_h n - 1 \tag{60}$$

secure AND operations. For B_s candidate batches,

$$C_{\text{col}} = B_s(J_h n - 1), \tag{61}$$

or asymptotically,

$$C_{\text{col}} = O(B_s J_h \log N) = O(B_s h^2 \log N). \tag{62}$$

All pairwise comparisons within a layer and all B_s candidate batches are evaluated in parallel. After collision detection, all batch-validity bits are opened in one batch. The resulting interaction depth is

$$\mathsf{rnd}_{\text{col}} = \lceil \log_2 n \rceil + \lceil \log_2 J_h \rceil + 1. \tag{63}$$

Since the validity pattern is public, selecting the first valid batch requires only public indexing.

F.3 One-Hot Support Expansion

After selecting a valid batch, each of its h indices is expanded into a length-N one-hot vector. For an index

$$j = (b_{n-1}, \dots, b_0), \tag{64}$$

the decoder starts from

$$V^{(0)}[0] = 1 \tag{65}$$

and recursively expands the active prefix. At layer ℓ, let $\beta_\ell = b_{n-1-\ell}$. For every current node,

$$R_k = V^{(\ell)}[m] \wedge \beta_\ell, \qquad L_k = V^{(\ell)}[m] \oplus R_m, \tag{66}$$

and

$$V^{(\ell+1)}[2m] = L_m, \qquad V^{(\ell+1)}[2m+1] = R_m. \tag{67}$$

The first expansion layer requires no secure AND because $V^{(0)}[0] = 1$ is public. For the remaining $n-1$ layers, the number of secure AND operations is

$$2 + 4 + \cdots + \frac{N}{2} = N - 2. \tag{68}$$

Hence, one decoder costs

$$N - 2 \tag{69}$$

secure AND operations, and all h decoders require

$$C_{\mathrm{OH}} = h(N-2). \tag{70}$$

The h decoders execute in parallel, while the decoder layers are sequential, giving

$$\mathsf{rnd}_{\mathrm{OH}} = n - 1 = \log_2 N - 1. \tag{71}$$

The positive and negative support indicators are formed by XORing the corresponding one-hot vectors and therefore require only local Boolean operations.

Combining support validation and one-hot expansion, the total sparse-secret Boolean work is

$$C_s = B_s(J_h n - 1) + h(N-2), \tag{72}$$

with interaction depth

$$\begin{aligned} \mathsf{rnd}_s &= \mathsf{rnd}_{\mathrm{col}} + \mathsf{rnd}_{\mathrm{OH}} \\ &= \lceil \log_2 n \rceil + \lceil \log_2 J_h \rceil + 1 + (n-1). \end{aligned} \tag{73}$$

F.4 Sparse-Sampling Failure Probability

A candidate batch consists of h indices sampled independently and uniformly from $[N]$. Its probability of being collision free is

$$p_{\mathrm{valid}} = \frac{N(N-1)\cdots(N-h+1)}{N^h} = \prod_{t=0}^{h-1} \left(1 - \frac{t}{N}\right). \tag{74}$$

Consequently, one candidate batch is invalid with probability

$$1 - p_{\mathrm{valid}}. \tag{75}$$

Since the B_s candidate batches are sampled independently, sparse sampling fails only if every batch contains a collision:

$$p^s_{\mathrm{fail}} = (1 - p_{\mathrm{valid}})^{B_s}. \tag{76}$$

Thus, it suffices to choose

$$B_s \geq \left\lceil \frac{(\lambda_{\mathrm{stat}} + 1)\ln 2}{-\ln(1 - p_{\mathrm{valid}})} \right\rceil \tag{77}$$

to obtain

$$p^s_{\mathrm{fail}} \leq 2^{-(\lambda_{\mathrm{stat}}+1)}. \tag{78}$$

Together with the dense-sampling bound

$$p^d_{\mathrm{fail}} \leq 2^{-(\lambda_{\mathrm{stat}}+1)}, \tag{79}$$

the union bound gives

$$\begin{aligned} p^{\mathrm{setup}}_{\mathrm{fail}} &\leq p^d_{\mathrm{fail}} + p^s_{\mathrm{fail}} \\ &\leq 2^{-\lambda_{\mathrm{stat}}}. \end{aligned} \tag{80}$$

F.5 RNS-CONSISTENT CONVERSION

Each secret s_x, for $x \in \{d, s\}$, is represented by two Boolean indicator vectors

$$\mathbf{P}_x, \mathbf{M}_x \in \{0,1\}^N, \tag{81}$$

where

$$s_x = \mathbf{P}_x - \mathbf{M}_x. \tag{82}$$

Converting one secret therefore requires $2N$ Boolean-to-RNS conversions and consumes exactly

$$2N \tag{83}$$

RNS-consistent correlated bits. Converting both s_d and s_s requires

$$4N \tag{84}$$

correlated bits.

For a Boolean-shared bit z, let r be an RNS-consistent correlated bit represented both as a Boolean sharing and as arithmetic shares across every modulus $q_\ell \in \mathcal{B}$. The protocol opens

$$\delta = z \oplus r. \tag{85}$$

Because r is uniform, δ is independent of z. For every q_ℓ, the arithmetic reconstruction satisfies

$$z \equiv \delta + (1 - 2\delta)r \pmod{q_\ell}. \tag{86}$$

Hence, the same opened mask is reused across all limbs while preserving cross-limb consistency.

All $4N$ masked Boolean values can be opened in a single batch. Therefore,

$$\mathsf{rnd}_{\mathrm{B2RNS}} = 1, \tag{87}$$

independent of

$$L_{\mathcal{B}} = |\mathcal{B}|. \tag{88}$$

The subsequent arithmetic-share computation is local. It requires

$$2NL_{\mathcal{B}} \tag{89}$$

constant-size arithmetic updates per secret and

$$4NL_{\mathcal{B}} = O(NL_{\mathcal{B}}) \tag{90}$$

for both secrets.

F.6 SSE-SPECIFIC KEY-MATERIAL COST

The cost of the conventional collective public key, relinearization keys, and Galois keys is inherited from the adopted threshold CKKS backend. Relative to this baseline, SSE introduces two cross-secret switching keys,

$$K^{qp}_{d\to s} \qquad \text{and} \qquad K^{QP}_{s\to d}. \tag{91}$$

Let L_{qp} and L_{QP} denote the numbers of RNS limbs in the corresponding modulus bases. If D_g denotes the gadget-decomposition length, then, up to backend-specific constants, the arithmetic work and key-material size of the two switching keys scale as

$$O(D_g N L_{qp}) \tag{92}$$

and

$$O(D_g N L_{QP}), \tag{93}$$

respectively. If different gadget-decomposition lengths are used for the two directions, the corresponding D_g terms are replaced by their direction-specific values.

The round complexity is inherited from the underlying threshold key-generation protocol. Importantly, both switching keys are generated from distributed shares of s_d and s_s. Neither collective secret is reconstructed during key generation.

Table 11: Early/Delayed ablation on the sparse-generation path.

Method	Conv (s)	Online (s)	Comm (MB)	Peak (MB)
Delayed	0.085	0.224	46.435	204.618
Early	1.947	2.201	707.039	1535.150
Early/Delayed	22.81×	9.82×	15.23×	7.50×

F.7 Ownership Handoff

In the concrete M-of-M instantiation, the final ownership handoff delivers one additive share from each auxiliary server to every party. Thus, each party receives two elements of R_Q.

Ignoring authentication tags, one element of R_Q contains N coefficients modulo Q and therefore requires

$$O(N \log Q) \tag{94}$$

bits. Across all M parties, the communication volume is

$$O(MN \log Q) \tag{95}$$

bits. The auxiliary servers perform

$$O(MN) \tag{96}$$

local coefficient operations, and all shares can be delivered in one communication round:

$$\mathsf{rnd}_{\mathrm{handoff}} = 1. \tag{97}$$

If the additive shares are subsequently converted to a t-of-M access structure, this final term is replaced by the communication and local-computation cost of the selected resharing backend. The preceding secret-generation and RNS-conversion costs are unchanged.

G Extra Evaluation

G.1 Extra Evaluation for RQ2

Boolean-First and Delayed-RNS Ablation. We evaluate delayed RNS realization and RNS-limb batching separately. *Early* realizes all h intermediate one-hot vectors in RNS, whereas *Delayed* first aggregates them in the Boolean domain and converts only the two final indicator vectors. The delayed variants batch all N coefficients and differ only in RNS-limb grouping: *Delayed* processes the limbs jointly, while *Per-Limb* processes them separately.

Delayed realization. We first quantify the benefit of delayed realization on the sparse-generation path at the target configuration. Table 11 shows that Early is $22.81\times$ slower in conversion. Early must realize all h intermediate one-hot vectors across the RNS basis, whereas Delayed aggregates them in the Boolean domain and converts only two final indicator vectors. The resulting intermediate-state expansion also increases communication and peak memory, explaining the larger online sparse-generation cost. Thus, the benefit comes from avoiding premature RNS materialization rather than changing the final secret representation.

Scaling sensitivity. We next vary h to determine whether the conversion overhead follows the size of the intermediate one-hot state. Early converts $hNL_{\mathcal{B}}$ bit–limb values, whereas both delayed variants convert only $2NL_{\mathcal{B}}$. Table 12 shows that Early conversion grows with h, while both delayed variants remain nearly unchanged because their logical workload is independent of h. The measured Early/Delayed latency ratios are $7.96\times$, $16.94\times$, and $33.63\times$, closely matching the corresponding workload ratios of $8\times$, $16\times$, and $32\times$. This confirms that delayed realization removes the h-dependent RNS expansion of intermediate one-hot state.

We further vary $L_{\mathcal{B}}$ at the target $h = 32$ to study RNS-basis scaling. Table 13 shows that all strategies become more expensive as the basis grows, while Early remains substantially slower because it realizes all intermediate vectors in RNS.

Table 12: h-scaling of RNS conversion at $N = 2^{16}$ and $L_{\mathcal{B}} = 40$. W_E/W_D denotes the Early-to-Delayed logical conversion-workload ratio.

h	W_E/W_D	Delayed (ms)	Early (ms)	Per-Limb (ms)
16	8×	379.530	3022.010	208.949
32	16×	373.217	6321.150	207.804
64	32×	381.340	12825.800	211.893

Table 13: RNS-basis scaling at $N = 2^{16}$ and $h = 32$.

$L_{\mathcal{B}}$	Delayed (ms)	Early (ms)	Per-Limb (ms)
4	27.127	578.019	26.395
8	63.202	1239.930	40.720
16	142.011	2336.680	88.857
40	416.049	6105.260	213.640

Limb-batching sensitivity. Finally, we isolate the effect of RNS-limb grouping. Delayed-40 and Per-Limb batch all N coefficients and perform the same $2NL_{\mathcal{B}}$ logical conversion work. Only the number of limbs processed per invocation differs. As shown in Table 14, under loopback, Per-Limb reduces total latency and peak memory by 12.1% and 28.3%, reflecting its smaller cross-limb working set. Under WAN latency, Delayed-40 instead reduces conversion and total latency by $1.192\times$ and $1.095\times$. Since communication is nearly identical, the reversal is mainly due to fewer serialized conversion interactions. Thus, limb batching trades working-set memory for interaction latency without changing the logical workload.

G.2 Extra Evaluation for RQ3

G.2.1 Distributed B2RNS Preprocessing Estimate.

The Direct-MPC implementation converts four N-entry Boolean indicator vectors, (P_d, M_d) and (P_s, M_s), into RNS shares. We estimate the correlation-material payload required by a fully distributed pairwise preprocessing scheme.

For each indicator bit, a party establishes an RNS-consistent correlation with each of the other $M-1$ parties. Hence, the number of RNS-vector correlations handled by one party is

$$T_{\text{vec}}^{\text{party}} = 4N(M-1). \tag{98}$$

Each RNS-vector correlation contains L limb-level correlations, giving

$$T_{\text{field}}^{\text{party}} = 4NL(M-1). \tag{99}$$

We model each limb-level pairwise correlation by two 64-bit residue words, i.e., 16 bytes of correlation material. The resulting per-party payload is therefore

$$B_{\text{B2RNS}}^{\text{party}} = 4NL(M-1) \cdot 16 = 64NL(M-1) \text{ bytes}. \tag{100}$$

Summed over all parties, the party-accounted aggregate payload is

$$B_{\text{B2RNS}}^{\text{agg}} = 64NLM(M-1) \text{ bytes}. \tag{101}$$

Thus, the modeled preprocessing traffic grows linearly with M per party and quadratically in aggregate.

For $N = 65536$ and $L = 40$,

$$B_{\text{B2RNS}}^{\text{party}} = 167{,}772{,}160(M-1) \text{ bytes}. \tag{102}$$

For example,

$$B_{\text{B2RNS}}^{\text{party}}(M=2) = 167{,}772{,}160 \text{ bytes} = 0.15625 \text{ GiB} = 0.167772 \text{ GB}, \tag{103}$$

$$B_{\text{B2RNS}}^{\text{party}}(M=64) = 10{,}569{,}646{,}080 \text{ bytes} = 9.84375 \text{ GiB} = 10.569646 \text{ GB}. \tag{104}$$

This estimate accounts for the pairwise RNS-consistent correlation material required by B2RNS.

Table 14: RNS-limb batching sensitivity at the target configuration. WAN uses 40-ms RTT/100 Mbps.

Network	Method	Offline (s)	Online (s)	Conv (s)	Total (s)	Comm (MB)	Peak (MB)
Loopback	Delayed-40	0.741	0.558	0.416	1.760	257.000	496.833
	Per-Limb	0.746	0.363	0.214	1.547	257.005	356.237
WAN	Delayed-40	10.437	18.639	14.688	29.568	257.000	480.518
	Per-Limb	10.412	21.462	17.502	32.380	257.005	356.684

Table 15: Contention-controlled contributor and auxiliary-server scaling. Co-located CPU reports the within-run median/maximum.

Parties (M)	Contributor-side		Auxiliary servers		
	Pinned (ms)	Co-loc. med/max (ms)	CPU (s)	Wall (s)	Handoff (s)
2	36.660	38.016/39.654	0.701	1.878	0.551
4	36.673	38.003/39.457	0.717	2.218	0.901
8	37.197	38.236/42.190	0.812	3.111	1.787
16	37.714	38.800/46.587	0.935	4.673	3.386
32	37.970	38.164/48.206	1.204	7.973	6.228
64	37.693	37.882/51.443	1.735	14.716	12.659

G.2.2 Contention-controlled Experiment

We run a contention-controlled experiment with disjoint physical-core groups for contributors and auxiliary servers and fixed auxiliary-server worker budgets.

Table 15 shows that pinned typical-party CPU changes by only 2.8% from $M = 2$ to $M = 64$, while the co-located median remains near 38 ms. The larger per-run maximum therefore mainly reflects order-statistic effects and co-location contention rather than higher computation for each contributor. In contrast, auxiliary-server CPU and wall time increase from 0.701 to 1.735 s and from 1.878 to 14.716 s, respectively. At $M = 64$, handoff accounts for 12.659 s, or 86.0%, of the wall time, identifying it as the dominant large-M cost. WAN conditions further amplify this communication cost, with 40.3-ms RTT and 90.3-Mbps goodput versus 1.14 ms and 944.6 Mbps under LAN.

G.2.3 Cryptographic-Parameter Scaling.

We fix $M = 16$ and evaluate $N \in \{2^{14}, 2^{15}, 2^{16}\}$ with $h = 32$ and the common secure $L_{\mathcal{B}} = 8$. We separately vary $h \in \{16, 64\}$ at $N = 2^{16}$ and $L_{\mathcal{B}} = 40$.

Table 16 shows different sensitivities to N and h. The $4\times$ increase in N raises per-party communication and CPU by $3.99\times$ and $3.23\times$, respectively, as the coefficient-wise RNS state grows with the ring dimension. Setup and H-RSS increase more slowly. In contrast, increasing h from 16 to 64 changes communication and H-RSS by less than 0.1%, with only a 5.9% setup increase, since the RNS-state dimensions remain unchanged.

Overall, RQ3 shows that contributor-side cost remains largely independent of the participant population. Per-party communication stays at 42.475 MB, while typical contributor CPU changes by only 2.8% from $M = 2$ to $M = 64$. The growing aggregate cost is concentrated at the auxiliary servers, mainly in handoff, while memory remains stable. Across cryptographic parameters, cost is more sensitive to N than to h.

Table 16: Cryptographic-parameter scaling at $M = 16$. Setup is the staged preprocessed projection and H-RSS is the auxiliary-server memory upper bound.

Config.	Party CPU (ms)	Setup (s)	Party comm. (MB)	H-RSS (MB)
$N = 2^{14}, L_{\mathcal{B}} = 8$	3.371	1.168	2.236	53.621
$N = 2^{15}, L_{\mathcal{B}} = 8$	5.662	1.471	4.464	74.298
$N = 2^{16}, L_{\mathcal{B}} = 8$	10.896	2.055	8.919	86.815
$h = 16, L_{\mathcal{B}} = 40$	47.337	8.418	42.470	192.401
$h = 64, L_{\mathcal{B}} = 40$	46.144	8.916	42.484	192.581