SHIM: SCALING HOMOMORPHIC GENERATIVE TRANSFORMER INFERENCE VIA MULTI-PARTY COLLABORATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Privacy-preserving large language model inference with homomorphic encryption remains impractical due to both latency and memory bottlenecks, particularly during autoregressive decoding. Existing non-interactive FHE systems incur high end-to-end latency from large-precision homomorphic operations and frequent bootstrapping, while hybrid HE-2PC designs reduce bootstrapping but introduce substantial communication overhead, especially for non-linear layers. Moreover, decoding requires a KV cache whose encrypted form leads to severe memory amplification. We propose *SHIM*, a collaborative framework that integrates threshold HE and secret sharing to enable a threshold-mediated masking paradigm. Our key insight is a hybrid representation of LLM state: linear layers remain encrypted, while KV cache and non-linear function inputs are maintained as cryptographically masked plaintexts. This design enables exact evaluation of Softmax, LayerNorm, and SiLU on masked values without polynomial approximations, simultaneously reducing both activation latency and bootstrapping overhead. Meanwhile, masked KV cache storage reduces memory from 198 GB to 3 GB compared to FHE baselines, enabling long-context decoding that exceeds memory limits in prior systems. *SHIM* achieves up to $15.7\times$ lower generation latency compared to recent state-of-the-art baselines, including NEXUS (NDSS'25), THOR (CCS'25), Powerformer (ACL'25), and BOLT (S&P'24), while preserving model utility.

1 INTRODUCTION

The rapid advancement of Large Language Models (LLMs) has transformed natural language processing, enabling unprecedented performance in text generation, reasoning, and complex problem-solving. As these models are increasingly deployed as cloud-hosted Machine Learning as a Service (MLaaS), they encounter a fundamental *privacy-utility stalemate*: user prompts or model outputs, often containing sensitive personal, financial, proprietary data, or regulated genomic records, are exposed to the service provider during computation. This exposure poses substantial privacy risks and creates significant barriers to adoption in security-, and enterprise-critical workloads, particularly in regulated domains like healthcare under HIPAA U.S. Congress (1996).

Homomorphic Encryption (HE) Gentry (2009) offers a principled cryptographic approach by enabling computation directly over encrypted ciphertexts. Despite advances in HE for linear algebra, existing HE-LLM systems Zhang et al. (2025); Moon et al. (2024); Park et al. (2025a) largely focus on the highly parallelizable *prefill stage*, with comparatively little attention paid to the **auto-regressive decoding stage**. This gap is unsurprising: decoding is inherently more challenging due to its *sequential, token-by-token nature*, which exposes two fundamental bottlenecks in the current HE-LLM paradigm:

- **The KV-Cache Memory Wall:** Efficient LLM decoding relies on the Key-Value (KV) cache to avoid redundant computation. In the HE domain, however, the KV cache must remain encrypted to preserve user privacy, resulting in severe memory amplification from ciphertext expansion. For example, for an 8K-token context in Llama-3.1-8B, an encrypted KV cache under encryption parameter $D = 2^{16}$ can grow to $\sim$198 GB, nearly $66\times$ larger

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.

than $\sim$3 GB plaintext KV cache, making long-sequence decoding almost infeasible on commodity GPUs.

- **The Nonlinear Approximation Wall:** Transformer decoding depends heavily on complex non-linear operations (e.g., SiLU, LayerNorm, Softmax). Since HE supports only arithmetic circuits, these operations are replaced with polynomial approximations using either (1) fitting polynomials on representative data Lee et al. (2022); Moon et al. (2024), or (2) fine-tuning the model with polynomial activations Lee et al. (2021); Tong et al. (2024). Both approaches require substantial training effort and generalize poorly. High-precision evaluation demands higher-degree polynomials, creating a *bottleneck of compounding inefficiency*: polynomial evaluation is costly, deeper circuits accelerate ciphertext noise growth, and frequent, slow *bootstrapping* (ciphertext refresh) is required. Combined, nonlinear evaluation and bootstrapping can account for over 82% of total inference latency Zhang et al. (2025).

In this paper, we break these bottlenecks by proposing *SHIM*, a framework centered on a novel paradigm. *Our core insight is to decouple the LLM's state into a hybrid representation: while linear projections remain in the encrypted domain, the KV cache and non-linear activations are maintained as cryptographically **masked plaintexts***. By leveraging a multi-party **Threshold HE and Secret Sharing (THE-SS)** protocol design, the server can evaluate exact nonlinear functions and perform attention lookups directly on masked data. This approach achieves significant inference speedup during decoding with almost zero loss on numerical parity. Furthermore, because the protocol implicitly resets ciphertext noise during the masking process, it provides a fast, low-cost refresh mechanism that bypasses the need for traditional HE bootstrapping entirely, finally making private long-sequence generation a practical reality. Our primary contributions are summarized as follows:

1. **Exact Non-Linearity via Secure Masking:** We design a hybrid protocol that offloads SiLU, LayerNorm, and Softmax to the plaintext domain via collaborative masked computation. By passing accuracy-degrading polynomial approximations, our approach achieves almost zero loss in numerical parity while eliminating the heavy computational overhead of high-degree HE circuits.
2. **Scalable Private KV Cache for Long Sequences:** We introduce a memory-efficient management scheme that stores the KV cache in a masked plaintext format rather than as full ciphertexts. This reduction, from prohibitive gigabyte-scale expansion (e.g. 198 GB) to near-plaintext levels, overcomes the primary memory bottleneck of private inference, enabling practical long-context decoding and larger batch processing on commodity GPUs.
3. **Efficient Threshold-based Noise Management:** We leverage the threshold nature of our protocol to perform "opportunistically refreshing" of ciphertexts during the evaluation of non-linear layers. This low-cost refresh mechanism bypasses the prohibitive latency of traditional FHE bootstrapping, fundamentally improving the end-to-end scalability of deep LLM architectures.
4. **Theoretical Rigor and Empirical Superiority:** We introduce novel additive and multiplicative masking schemes and derive formal **perturbation bounds** to theoretically guarantee the security and numerical stability of the system. Our evaluation demonstrates that *SHIM* provides a significant end-to-end speedup, outperforming several recent state-of-the-art frameworks, including NEXUS (NDSS'25) Zhang et al. (2025) and THOR (CCS'25) Moon et al. (2024), by up to 15.7$\times$ in generation latency while scaling to significantly longer sequences.

2 PRELIMINARY

Homomorphic Encryption—CKKS Homomorphic Encryption allows computations on encrypted data. HE has different categories according to the different computation types they support. The Leveled HE (LHE) schemes support a limited number of additions or multiplications while Fully HE (FHE) allows an arbitrary number of computations using a bootstrapping procedure that can effectively refresh the ciphertext and obtain a new ciphertext that encrypts the same value but has lower noise Gentry (2009). CKKS Cheon et al. (2017) is an FHE scheme and its security is based on the hardness of ring learning with errors (RLWE) problem. For more details of CKKS, refer to Appendix B.

LLM and KV Cache. Autoregressive Transformer LLMs generate tokens sequentially and use a KV cache to avoid recomputing attention over previous tokens. We use standard notation for token generation, attention, and cache updates; the equations and details are in Appendix F.

Related Works. Existing HE based secure inference systems can be broadly grouped into two categories. The first category relies on FHE in a non interactive setting Gilad-Bachrach et al. (2016); Chou et al. (2018); Jiang et al. (2018); Brutzkus et al. (2019); Dathathri et al. (2019); Kim et al. (2022a); Lee et al. (2022); Ran et al. (2023); Zhang et al. (2025). These approaches enable deep circuit evaluation but typically depend on bootstrapping to control ciphertext noise, which remains a major source of end to end latency Kim et al. (2021); Lee et al. (2022); Kim et al. (2022b); Jung et al. (2021). The second category adopts hybrid protocols that combine HE with two party computation or secret sharing Juvekar et al. (2018); Mishra et al. (2020); Huang et al. (2022); Li et al. (2024); Pang et al. (2024); Lu et al. (2023); Xu et al. (2025). By converting intermediate values into secret shared form for decryption and re encryption, these methods can avoid bootstrapping and often reduce the cost of secure linear layers compared to pure MPC protocols Koti et al. (2021); Dalskov et al. (2021); Kumar et al. (2020); Mohassel & Rindal (2018); Mohassel & Zhang (2017). However, their performance is frequently limited by communication, especially for handling non linear functions.

Recent FHE based systems further improve accuracy by using high order polynomial approximations for transformer non linearities, such as NEXUS (Zhang et al., 2025), THOR Moon et al. (2024), and Powerformer Park et al. (2025a). While effective, high order polynomials increase the number of homomorphic operations and the multiplicative depth, which in turn exacerbates bootstrapping overhead. More fundamentally, polynomial approximation does not resolve the non linear evaluation challenge in HE, since achieving acceptable accuracy typically requires dataset specific calibration, such as range profiling, approximation fitting, and or additional fine tuning, which introduces extra training cost and limits generality across datasets and tasks. Additionally, several model-wise optimizations are complementary to our work. For example, BLB Xu et al. (2025) reduces communication via circuit level fusion in hybrid protocols, and MPCache Zeng et al. (2025) studies KV cache eviction under encryption. These directions can be combined with our approach to further reduce the computational overhead.Cachemir Yu et al. (2026) improves FHE-based LLM inference through KV-cache-aware packing and bootstrapping placement, whereas *SHIM* focuses on threshold masking for non-polynomial computation and masked KV states; the two directions are complementary. Recent work studies provable security for CKKS-based MPC, including stronger simulation-oriented treatments of smudging noise Hwang et al. (2025). This line is complementary to *SHIM*: our current protocol and evaluation target a semi-honest, honest-majority deployment, so the additional risk scenarios and protocol machinery in that work are outside our present scope.

3 System Model

3.1 Architecture Overview

Our system consists of N_S servers $\{S_i\}_{i=1}^{N_S}$ that collectively provide secure computation, connected via high-bandwidth network and operated by distinct entities to ensure no single entity controls multiple servers. For a client session assigned to S_i, S_i acts as the computation server and executes HE operations on the encrypted query, while the other $N_S - 1$ servers ($\{S_j\}_{j=1,j\neq i}^{N}$) act as the peer servers assist it through threshold cryptographic operations. To support these roles, each server S_i can be equipped with $\mathcal{G}_i$ GPU nodes logically partitioned into *computation nodes* $\mathcal{P}_i$ (used when S_i is a computation server) and *helper nodes* $\mathcal{H}_i$ (used to assist peer computation servers), with $|\mathcal{P}_i| + |\mathcal{H}_i| = \mathcal{G}_i$. This partition is configurable based on workload, with the key principle that $\mathcal{H}_i$ operates independently from $\mathcal{P}_i$ to prevent computation pipeline stalls.

Our architecture supports parallel processing of encrypted queries when multiple servers are authorized by the model owner to act as computation servers. In that optional deployment, each authorized computation server hosts a synchronized model copy, clients can route independent sessions to different servers, and aggregate throughput increases with the number of computation servers. The model-copy replication and multi-server parallelism are serving choices for throughput, not requirements of *SHIM*: the cryptographic protocol can also operate with one model-hosting computation server and other servers acting only as helpers. In either deployment, computation

servers cooperatively use peer (helper) servers during noise refresh, masked nonlinear operations, and KV cache management, etc., while disjoint $\mathcal{P}_i$ and $\mathcal{H}_i$ resources can prevent pipeline stalls.

When the optional parallel deployment is used, the model owner distributes synchronized copies of a pre-defined LLM only to the servers authorized to act as computation servers. This enables interchangeable query routing and load balancing across those computation servers.

3.2 Multi-Key Architecture

In our system, we consider *per-client* key pairs generated on-demand for each client session. Let $1 \leq N_{\text{comp}} \leq N_S$ denote the number of servers authorized by the model owner to act as computation servers in a given deployment. Each such server can host up to N_C active client sessions simultaneously, with each client assigned to one primary computation server for query processing. A resource scheduler handles load balancing across the authorized computation servers. In the optional fully parallel deployment, the cluster supports up to $N_{\text{comp}} \times N_C$ concurrent clients; ***SHIM*** itself does not require $N_{\text{comp}} = N_S$, and a single computation server can use the remaining peer servers as helpers. Compared to standard threshold encryption schemes that use a single shared key, our per-client key design provides strong session isolation: When a client session c is assigned to server S_i, S_i does not hold any share of that client's secret key $sk^{(c)}$. Specifically, $sk^{(c)}$ is split through additive secret sharing among the $N_S - 1$ peer servers $\{S_j\}_{j=1,j\neq i}^{N_S}$. Here client session refers to a time-bound interaction period during which one client uses the system for encrypted inference. The details of additive secret sharing is introduced in Appendix D.

To enable threshold encryption, we employ a trusted third-party key generator to create fresh cryptographic material for each client session. This is a standard trusted dealer assumption in threshold cryptography Desmedt (1993); Canetti (2020). As a result, S_i obtains the public key $pk^{(c)}$ and evaluation keys $\mathbf{evk}^{(c)}$. The peer servers S_j hold the secret key share $sk_j^{(c)}$ and client gets the secret key sk for decrypting the final result. We refer the reader to Appendix C for details of the key generation and distribution.

3.3 Threat Model

We consider N_S servers $\{S_i\}_{i=1}^{N_S}$ that collectively provide a secure computation service. We consider a *static semi-honest* (honest-but-curious) adversary that may corrupt at most $N_S - 2$ servers (Here we denote by $\mathcal{A}$ the coalition of servers corrupted by the adversary); this is the privacy threshold of the per-round additive sharing, not an honest-majority claim for arbitrary N_S. Malicious or adaptive corruption and implementation side channels are outside our model. Our system employs *per-client session keys* where in each client session c, the client holds the unique and complete secret key $sk^{(c)}$ for decrypting inference results, while the peer servers collectively hold additive shares of that key. During a masked-decryption round, the computation server receives masked plaintext and each peer retains its fresh mask share. Under additive sharing, any coalition of up to $N_S - 2$ corrupted servers learns no information about the secret key, providing $(N_S - 2)$-privacy. Per-client independent keys provide session isolation: compromise of one session's key material does not by itself expose other sessions; protection of completed sessions assumes secure erasure. Our security goals include input privacy (confidentiality of plaintext queries and results, subject to the leakage profile in Appendix L), KV-cache privacy (mask shares remain distributed among peers and are not reconstructed by the computation server), and refresh confidentiality (the computation server receives masked decryption results). We assume a trusted, non-colluding key generator that operates during system runtime, client-side secure key storage, and authenticated encrypted channels between all servers.

4 Protocol Overview

Our framework orchestrates homomorphic computations and threshold refresh based on runtime conditions, replacing expensive bootstrapping with efficient threshold decryption while opportunistically exploiting masked plaintexts to compute non-linear operations in plaintext domain.

More specifically, during inference on server S_i for client session c, the *masked threshold refresh* protocol is triggered when either of the following conditions is satisfied: (1) the ciphertext's remaining

noise budget reaches zero, making refresh *mandatory* to restore the noise budget and enable continued homomorphic computation; or (2) the immediately following computation involves a non-linear function (e.g., SiLU activation, LayerNorm variance, or Softmax) that is prohibitively expensive to evaluate homomorphically, as it requires high-degree polynomial approximations or complex circuits. Then, as the first step of the protocol, S_i sends its current ciphertext $ct = (c_0, c_1) = \text{Enc}_{pk^{(c)}}(X)$ where X is the corresponding encoded plaintext, to the peers $\{S_j\}_{j=1,j\neq i}^{N}$. The peers then conduct the partial decryption by individually applying their secret key shares $\{sk_j^{(c)}\}_{j=1,j\neq i}^{N_S}$ on ct, as

$$d_j = \text{PDec}_{sk_j^{(c)}}(ct) = c_1 \cdot sk_j^{(c)} + e_{sm,j} \mod Q, \text{ with } sk^{(c)} = \sum\nolimits_{j\in\mathcal{T}_i} sk_j^{(c)} \mod Q \ , \quad (1)$$

where $\mathcal{T}_i = \{1, \ldots, N_S\} \setminus \{i\}$. Note here to protect the partially decrypted data against the secret key share ($sk_j^{(c)}$) recovery attacks, as introduced in Li et al. (2023), we also add the *smudging* noise Li et al. (2023); Asharov et al. (2012); Froelicher et al. (2023) into the partial decryption, where $e_{sm,j}$ is the random noise sampled from the discrete Gaussian distribution with the variance is Δ Li et al. (2023) (same to the scale factor). Next, we detail the subsequent steps for the two cases: when the ciphertext's noise budget has been exhausted and when a nonlinear operation is forthcoming. We also summarize the notation used throughout the paper in Appendix A.

4.1 Noise Budget is Exhausted ($\mathbf{P}_{\text{REFRESH}}$)

We define ℓ_{ct} as the ciphertext's current noise level. When the peers $\{S_j\}_{j=1,j\neq i}^{N_S}$ (threshold parties) receive a ciphertext whose noise budget has been exhausted ($\ell_{ct} = 0$), after applying partial decryption as in Eq. (1), each threshold party S_j creates an additive mask M_j and adds it on the partial decryption d_j, which yields a masked partial decryption $d_j + M_j$. S_j then sends the masked partial decryption along with the encrypted mask $\text{Enc}_{pk^{(c)}}(M_j)$, back to S_i. At S_i's end, once $N_S - 1$ masked partial decryptions are collected, S_i securely combines all shares, which results in a masked plaintext

$$X + M = c_0 + \sum\nolimits_{j\in\mathcal{T}_i}(d_j + M_j) \mod Q \ , \quad (2)$$

Algorithm 1 *SHIM* Protocol

Input: Client session c assigned to server S_i, peer servers $\{S_j\}_{j\in\mathcal{T}_i}$, layered computation circuit $\mathcal{C} = \{\mathcal{C}_l\}_{l=1}^{L}$, maximum noise level $\ell_{\max}$, token limit $\tau_{\max}$

Preprocessing: $\{pk^{(c)}, \mathbf{evk}^{(c)}, sk^{(c)}\}$ and $\{sk_j^{(c)}\}_{j\in\mathcal{T}_i}$ are generated and distributed correspondingly (Appendix C), input prompt is encrypte $ct_{\text{input}} = \text{Enc}_{pk^{(c)}}(X_{\text{input}})$
Initialization: $\ell_{ct} \leftarrow \ell_{\max}$, $t \leftarrow 0$, sequence $\mathbf{a}_{ct} \leftarrow ct_{\text{input}}$, sequence $Y_{\text{output}} = \emptyset$
While $t < \tau_{\max}$ **and** not $\langle\text{EOS}\rangle$ **do**
    $t \leftarrow t + 1$;
    $ct \leftarrow \mathbf{a}_{ct}$;
  For each layer $l = 1$ to L
    For each circuit operation op_c in topological order of $\mathcal{C}_l$
      If ℓ_{ct} == 0
        Call noise refresh protocol (Sec. 4.1) to refresh ct: $ct = \mathbf{P}_{\text{refresh}}(ct)$; ($\ell_{ct} \leftarrow \ell_{\max}$)
      Else if op_c is non-linear ($op_c \in$ {Softmax, LayerNorm, SiLU})
        Call masked non-linear evaluation protocol (Sec. 4.2) to evaluate ct: $ct \leftarrow \mathbf{P}_{\text{nonlinear}}(ct, op_c)$;
      Else if op_c is linear
        If $op_c \in$ {attention-logit computation, attention-output computation}
          Call ciphertext–masked-plaintext product protocol (Sec. 4.3) to evaluate ct:
            $ct \leftarrow \mathbf{P}_{\text{unmaskmul}}(ct,\ Y_{op_c} + M_{Y_{op_c}})$; ($Y_{op_c}$ and $M_{Y_{op_c}}$ are specified in Appendix F)
        Else
          $ct \leftarrow op_c(ct)$;
        End
      End

      If new KV cache matrices $(\text{Enc}_{pk}(K_t^{(l)}), \text{Enc}_{pk}(V_t^{(l)}))$ are generated
        Call distributed masked KV cache management protocol (Sec. 4.3):
          $(K_{1:t}^{(l)} + M_{K,1:t}^{(l)}, V_{1:t}^{(l)} + M_{V,1:t}^{(l)}) \leftarrow \mathbf{P}_{\text{KVM}}(\text{Enc}_{pk}(K_t^{(l)}), \text{Enc}_{pk}(V_t^{(l)}))$;
      End
    End
  End
  $Y_{\text{output}} \leftarrow \{Y_{\text{output}}, ct\}$, $ct \leftarrow \{\mathbf{a}_{ct}, ct\}$;
End
Output: Encrypted output Y_{output}

where $X = c_0 + \sum_{j \in \mathcal{T}_i} d_j$ and $M = \sum_{j \in \mathcal{T}_i} M_j$. The generation and properties of the additive masking M would be discussed in detail in Appendix H.1. At S_j, M_j is generated according to the distribution of M scaled by $1/(N_S - 1)$. Furthermore, as $\text{Enc}_{pk^{(c)}}(M) = \sum_{j \in \mathcal{T}_i} \text{Enc}_{pk^{(c)}}(M_j)$ has also been collected at S_i, by removing the encrypted mask from the masked plaintext through $\text{Enc}_{pk^{(c)}}(X) = X + M - \text{Enc}_{pk^{(c)}}(M)$, the ciphertext of X is restored at the refreshed level $\ell_{\max}$ ($\ell_{ct} = \ell_{\max}$). Eventually, the refreshed $\text{Enc}_{pk^{(c)}}(X)$ is forwarded to the downstream layers in S_i for further computation. We denote by $\mathbf{P}_{\text{refresh}}$ (Alg. 2) the *noise refresh* protocol described above.

4.2 Non-linear Computation is Upcoming ($\mathbf{P}_{\text{nonlinear}}$)

Non-linear functions such as Softmax, LayerNorm, and SiLU are a major bottleneck in HE-based inference, since they cannot be evaluated efficiently in the encrypted domain and typically require polynomial approximations or expensive circuit constructions. Our framework addresses this by evaluating non-linear functions in the plaintext domain using masking, which preserves privacy while significantly reducing computation. Unlike Sec. 4.1 where noise refresh is *mandatory*, refresh here is triggered *opportunistically*: even if the noise budget remains *adequate*, the system triggers threshold refresh to exploit the masked plaintext for efficient computation.

Specifically, for client session c, instead of directly applying a non-linear operator to an encrypted value $\text{Enc}_{pk^{(c)}}(X)$, S_i invokes the collective decryption protocol to obtain a masked plaintext $X + M$ as in Eq. (2). However, unlike the noise refresh in Sec. 4.1 where S_i immediately removes the encrypted mask to reconstruct a refreshed ciphertext, the key insight here is that S_i applies the non-linear operator to the masked plaintext and only then compensates for the mask to recover the desired encrypted result. More specifically, S_i first computes $\phi(X+M)$ in plaintext, where ϕ denotes the non-linear operation (e.g., SiLU, Softmax), and obtains the encrypted result $\text{Enc}_{pk^{(c)}}(\phi(X+M))$. Since ϕ is non-linear and does not satisfy $\phi(X + M) = \phi(X) + \phi(M)$, we develop protocols to remove the mask's contribution by applying correction terms $\text{Enc}_{pk^{(c)}}(g(M_j))$ computed by each S_j with $j \in \mathcal{T}_i$, where function g depends on the specific operation ϕ. Eventually we could recover $\text{Enc}_{pk^{(c)}}(\phi(X))$. These operator-specific techniques are detailed in Appendix I.

While *additive* masking $X + M$ suffices for some transformer operations (such as Softmax), certain operations require *multiplicative* masking $X \cdot M$, which is essential for operations involving division or reciprocals. In multiplicative masking, after S_i broadcasts the plaintext to the peers, each peer S_j samples a non-zero mask M_j and computes the partial decryption $X_j = c_0 + d_j$. The objective is to compute the masked plaintext in the form of

$$X \cdot M = (\sum\nolimits_{j \in \mathcal{T}_i} X_j) \cdot (\sum\nolimits_{j \in \mathcal{T}_i} M_j) \ , \tag{3}$$

where the aggregated mask is $M = \sum_{j \in \mathcal{T}_i} M_j$. Typically, Eq. (3) *cannot* be obtained by *local multiplication* alone while maintaining *privacy*. Having each peer compute only $X_j M_j$ locally and send it to S_i fails because $\sum_{j \in \mathcal{T}_i}(X_j M_j)$ includes only diagonal terms and does not equal XM. More specifically, the right side of Eq. (3) requires all cross-terms $X_h M_k$ for $h \neq k$, hence each peer needs other peers' shares. One might consider generating identical M_j across all threshold parties to satisfy Eq. (3) via local computation alone. However, this clearly compromises security: *any* single corrupted peer would learn the full mask M, enabling it to recover X from the masked plaintext.

Hence, we adopt standard MPC techniques designed for additive secret sharing to securely compute $X \cdot M$. Specifically, the peer servers hold additive shares $[[X]] = \{X_j\}_{j \in \mathcal{T}_i}$ and $[[M]] = \{M_j\}_{j \in \mathcal{T}_i}$, where $[[\cdot]]$ denotes the additive secret sharing scheme. Peers execute a secure multiplication protocol $\otimes$ Damgård et al. (2013) to compute shares $[[Z]] = [[X]] \otimes [[M]]$ where $Z = X \cdot M$. The MPC protocol ensures that each peer obtains a result share Z_j without any peer learning X, M, or Z during the computation. Finally, each peer S_j sends its result share Z_j and operation-specific encrypted mask correction term $\text{Enc}_{pk^{(c)}}(g(M_j))$ to S_i, who reconstructs the masked plaintext through

$$X \cdot M = \sum\nolimits_{j \in \mathcal{T}_i} \{Z_j\} \ , \tag{4}$$

computes the non-linear function $\phi(XM)$ in plaintext, and then obtains $\text{Enc}_{pk^{(c)}}\phi(XM)$. Eventually, S_i recovers $\text{Enc}_{pk^{(c)}}(\phi(X))$ by removing the mask's contribution with the aid of $\{\text{Enc}_{pk^{(c)}}(g(M_j))\}_{j \in \mathcal{T}_i}$. We refer readers to Appendix I for further details on the masked non-linear evaluations. The generation of the multiplicative mask M_j follows the distribution of M H.2 scaled

by a factor of $1/(N_S - 1)$. We use $\mathbf{P}_{\text{nonlinear}}$ (Alg. 3) to denote the *masked non-linear computation* protocol described above.

4.3 KV Cache Management ($\mathbf{P}_{\text{KVM}}$) and Ciphertext–Masked-Plaintext Product ($\mathbf{P}_{\text{UNMASKMUL}}$)

We use two protocols for KV-cache handling: $\mathbf{P}_{\text{KVM}}$ converts newly generated encrypted key/value vectors into masked plaintext cache entries, and $\mathbf{P}_{\text{unmaskmul}}$ computes encrypted products with the masked cache without reconstructing it in the clear. Appendix F gives the cache-update and attention-computation details.

End-to-End Workflow. We also summarized our *SHIM* protocol in Alg. 1. Here a layered *computation circuit* $\mathcal{C} = \{\mathcal{C}_l\}_{l=1}^{L}$ representing the transformer architecture (L layers in total) as a directed acyclic graph (DAG) over the transformer's L layers, where C_l is the set of circuit operations in layer l, and each node $opc \in C_l$ is categorized as linear (the linear nodes are further split into ordinary linear operations and those requiring a ciphertext–masked-plaintext product), or non-linear (including Softmax, LayerNorm, SiLU, deatiled evaluation see in Appendix K), augmented with conditional refresh triggers. X_{input} is the input prompt, $\tau_{\max}$ is the token limit and Y_{output} is the final output under encryption, which is further decrypted by the client using $sk^{(c)}$. Additionally, with our *SHIM*, we do not require model alteration for non-linear functions compared to previous methods Xie et al. (2026); Peng et al. (2023).

4.4 Security Analysis

Theorem L.5 in Appendix L summarizes the security guarantees; detailed analyses and proofs are also provided in Appendix L. We refer the reader there for a comprehensive discussion.

5 Evaluation

5.1 Experimental Setup

Benchmarks. To evaluate our hybrid framework, we benchmark the system on the modern transformer architecture LLaMA-3.1-8B-Instruct Touvron et al. (2023) using LongBench Bai et al. (2024), which includes multiple sub-tasks. We test with KV cache budgets of 4096, 8192, 16384, and 32768 tokens, respectively. Our collaborative cloud environment consists of $N_S = 3$ servers: for each server processing client queries, the other 2 servers act as threshold parties and participate in the three-party secure multiplication protocol for multiplicative masking. We conduct all experiments under a WAN setting (100 Mbps bandwidth, 40 ms latency) to assess the framework's robustness against network conditions.

Comparative Methodology and Baselines. To the best of our knowledge, state-of-the-art HE-based frameworks, including interactive HE-2PC system *BOLT* (S&P'24) Pang et al. (2024) and non-interactive FHE systems *NEXUS* (NDSS'25) Zhang et al. (2025), *Powerformer* (ACL'25) Park et al. (2025a), and *THOR* (CCS'25) Moon et al. (2024), are primarily designed for the parallelizable prefill stage and do not provide native support for the sequential decoding stage or the associated KV cache memory growth. To ensure rigorous and fair comparison, we extend these baselines by implementing an optimized CPU-GPU memory swapping strategy for their encrypted KV caches, enabling us to evaluate their end-to-end decoding performance across context lengths (4K to 32K)

Table 1: Comparison with SOTA methods on LLaMa-3.1-8B with 8192 KV-cache budget.

Method	Activation Latency (s)	BOOT Latency (s)	Total Latency (s)	Speedup	KV-cache Memory (GB)	Communication (GB)
NEXUS (NDSS'25)	34.6	120.9	213.9	-	198	-
THOR (CCS'25)	2.3	14.0	41.8	5.1×	198	-
Powerformer (ACL'25)	0.4	4.3	30.1	7.1×	198	-
BOLT (S&P'24)	57.8	-	72.3	2.9×	25	0.43
Ours	0.002	-	13.7	15.7×	3	0.07

within our hardware limits. Critically, we include all CPU-GPU transfer latencies in our end-to-end measurements for these extended baselines, revealing the I/O-compute bottleneck inherent in pure FHE approaches—a bottleneck our protocol fundamentally bypasses by maintaining the entire working state within GPU memory.

Encryption Parameters. For the HE scheme, we use CKKS encryption parameters with $D = 2^{16}$ and $\Delta = 2^{40}$ to support the large-scale computations required by LLMs. Non-linear operations (Softmax, LayerNorm, and SiLU) are evaluated using our masked-plaintext protocol, aligning with standard PyTorch implementations. For secure multiplication in MPC, we use the protocol from Araki et al. (2016) to instantiate the multi-party multiplication building blocks and implement them in MP-SPDZ Keller (2020). For additive and multiplicative masking, we select $R = 2^{30}$ and K=2, respectively.

Hardware Environment. All experiments are conducted on a workstation with an AMD Threadripper PRO 7975WX, 512 GB RAM, and a single NVIDIA RTX A6000 (48 GB VRAM) connected via PCIe 4.0 $\times$16 (29–30 GB/s runtime bandwidth, 16–18 μs one-way latency). We use the CUDA backend of Liberate.FHE v0.9.0 DESILO (2023) for GPU-accelerated RNS-CKKS Cheon et al. (2018) implementation.

5.2 Result Analysis

End-to-End performance against SOTA Work. We first compare end to end latency and communication size against SOTA baselines, including Bolt (S&P'24), and non-interactive protocols like NEXUS (NDSS'25), Powerformer (ACL'25), and THOR (CCS'25). We select LLaMA 3.1-8B with an 8192 KV-cache budget as the evaluation benchmark.

As shown in Table 1, our hybrid framework achieves substantial improvements in total latency and communication efficiency compared to state-of-the-art baselines. It achieves a total latency of 13.7 seconds, outperforming NEXUS (213.9s), THOR (41.8s), Powerformer (30.1s), and BOLT (72.3s), representing up to 15.7$\times$ speedup over NEXUS. Moreover, our method requires only 3 GB of KV cache memory, compared to 198 GB for fully-encrypted solutions (NEXUS, THOR, and Powerformer). These performance gains primarily stem from our threshold HE and secret sharing protocol, which eliminates expensive high-degree polynomial approximations, avoids encrypted KV cache storage, and replaces bootstrapping in the large-bit HE domain with efficient operations in the low-bit plaintext domain, greatly reducing both computation and storage overhead.

Among all components, the most significant improvement comes from non-linear layers. Our activation latency is only 0.002 seconds, compared to 34.6 seconds in NEXUS, 2.3 seconds in THOR, 0.4 seconds in Powerformer, and 57.8 seconds in BOLT. This speedup stems from our protocol design, which avoids both major sources of overhead in prior approaches. Compared to FHE systems, we do not rely on high-degree polynomial approximations for Softmax, LayerNorm, and SiLU, thereby avoiding the large multiplicative depth and frequent bootstrapping operations. Compared to hybrid HE-2PC approaches, our method does not require highly interactive protocols that incur heavy communication for non-linear evaluation. As a result, total communication volume is reduced to 0.07 GB versus 0.43 GB in BOLT, demonstrating that masked-plaintext-based non-linear evaluation achieves low latency without substantial communication overhead.

Additionally, by eliminating the encrypted KV cache, our method reduces KV cache storage from 198 GB in FHE-based solutions to 3 GB. This substantial reduction enables larger-scale inputs and longer context lengths in practice. We further validate these scalability benefits in Sec. 5.2.

Table 2: Error bounds of approximated nonlinear operators and downstream F1

Dataset	MAE			F1 score	
	softmax	SiLU	LayerNorm	Ours	Original
triviaqa	1.1×10^{-3}	8.0×10^{-4}	6.5×10^{-4}	91.28	91.61
hotpotqa	1.5×10^{-3}	1.0×10^{-3}	8.0×10^{-4}	56.89	57.33
2wikimqa	1.6×10^{-3}	1.1×10^{-3}	9.0×10^{-4}	47.16	47.81
qasper	1.7×10^{-3}	1.2×10^{-3}	9.5×10^{-4}	46.63	47
musique	1.9×10^{-3}	1.3×10^{-3}	1.0×10^{-3}	31.59	32.25
narrativeqa	2.0×10^{-3}	1.4×10^{-3}	1.1×10^{-3}	29.63	30.05

Accuracy and utility under our protocol. Table 2 evaluates the fidelity of our masked-plaintext-based non-linear execution by reporting the mean absolute error (MAE) of Softmax, SiLU, and LayerNorm, along with downstream F1 scores relative to the plaintext model. A key advantage of our design is that it preserves the original functional form of these operators rather than replacing them with polynomial surrogates, thereby avoiding approximation-induced utility loss that can accumulate across layers and decoding steps. As shown in Table 2, the operator-level errors remain small with MAE around 10^{-3}, and downstream performance closely matches the plaintext baseline across all tested datasets. For example, we achieve 91.28 F1 on TriviaQA compared to 91.61 for plaintext. We attribute this small gap not to the non-linear function computation but rather to the limited KV cache budget (8K tokens), as the baseline assumes unlimited cache Park et al. (2025b).

Table 3: Scalability comparison (maximum batch size and latency per token) under different KV-cache budget length. OOM refers to out of GPU memory.

KV cache budget	Method	Maximum batch size	OOM	Latency/Token
4K	BOLT	4	✗	73.3
	Ours	32	✗	11.2
8K	BOLT	2	✗	72.2
	Ours	16	✗	12.7
16K	BOLT	-	✓	-
	Ours	8	✗	13.1
32K	BOLT	-	✓	-
	Ours	4	✗	13.4

Scalability Analysis In practical LLM serving, the primary goal is to performing auto regressive decoding, where the KV cache is essential. Our framework resolves the encrypted KV cache memory bottleneck by storing the cache as masked plaintexts rather than full ciphertexts, which substantially improves scalability with respect to both context length and batch size. Table 3 shows that our method supports significantly larger batches at the same KV cache budget: at 4K context length, the maximum batch size increases from 4 (BOLT) to 32, reducing per-token latency by $6.5\times$; at 8K, it increases from 2 to 16, achieving a $5.7\times$ reduction. Moreover, our method continues to scale to longer context lengths where ciphertext-based caching becomes infeasible. While BOLT encounters out-of-memory errors at 16K and 32K, our system successfully supports 16K with batch size 8 and 32K with batch size 4. Importantly, our per-token latency also decreases as batch size grows, reducing from 13.4 s at batch size 4 to 11.2 s at batch size 32. This is because batched inputs in the HE domain reduce per-token computation overhead through more efficient rotation usage Ran et al. (2022). This demonstrates that our KV cache management not only reduces memory but also improves computation latency. Overall, these results show that masked-plaintext KV caching enables long-context autoregressive generation under encryption with substantially improved throughput and robust latency scaling.

6 Conclusion

In this paper, we presented *SHIM*, a hybrid protocol that addresses the key latency and memory bottlenecks in privacy-preserving LLM inference. By leveraging a collaborative cloud architecture with threshold homomorphic encryption and secret sharing, our framework overcomes core limitations of prior FHE-only and HE-MPC designs, demonstrating that practical low-latency private inference is achievable for large-scale transformers, especially in the autoregressive decoding setting. At the algorithmic level, our **masked-plaintext evaluation** protocol enables exact evaluation of computationally expensive non-linear operations, including Softmax, LayerNorm, and SiLU, without relying on polynomial approximations. This avoids approximation-induced utility loss and substantially reduces the non-linear evaluation overhead that dominates prior systems. At the system level, we introduce a privacy-preserving KV cache management strategy for HE-based inference. By storing KV cache entries as masked plaintexts rather than ciphertexts, we reduce the KV cache memory footprint for LLaMA-3.1-8B at 8K context from $\sim$198 GB to 3 GB, improving storage efficiency by $66\times$ and enabling scalable long-context generation up to 32K context length.

AI USE STATEMENT

We used generative AI tools primarily to polish the manuscript's language and improve clarity. The authors reviewed the AI-assisted text and take responsibility for the final content.

ETHICS STATEMENT

This work develops techniques for privacy-preserving inference using publicly available models and benchmarks. It does not recruit human subjects or collect new personal data; we support responsible deployment that protects user privacy.

REPRODUCIBILITY STATEMENT

The main text and Appendix describe our protocols, security assumptions, and analysis. Section 5.1 specifies the benchmark, model, cache budgets, server and network configuration, cryptographic parameters, and hardware/software used in the evaluation to support reproduction of the reported results.

REFERENCES

Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. High-throughput semi-honest secure three-party computation with an honest majority. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS'16)*, pp. 805–817, 2016.

Gilad Asharov, Abhishek Jain, Adriana López-Alt, Eran Tromer, Vinod Vaikuntanathan, and Daniel Wichs. Multiparty computation with low communication, computation and interaction via threshold fhe. In *Proceedings of the 31st Annual International Conference on Theory and Applications of Cryptographic Techniques (EUROCRYPT'12)*, EUROCRYPT'12, pp. 483–501, 2012.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, et al. Longbench: A bilingual, multitask benchmark for long context understanding. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pp. 3119–3137, 2024.

Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. (leveled) fully homomorphic encryption without bootstrapping. *ACM Transactions on Computation Theory (TOCT)*, 6(3):1–36, 2014.

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

Ran Canetti. Universally composable security. *Journal of the ACM*, 67(5), September 2020. ISSN 0004-5411.

Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Jeffrey Hoffstein, Kristin Lauter, Satya Lokam, Dustin Moody, Travis Morrison, et al. Security of homomorphic encryption. *HomomorphicEncryption. org, Redmond WA, Tech. Rep*, 2017.

Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. Homomorphic encryption for arithmetic of approximate numbers. *Proceedings of Advances in Cryptology - ASIACRYPT*, pp. 409–437, 2017.

Jung Hee Cheon, Kyoohyung Han, Andrey Kim, Miran Kim, and Yongsoo Song. A full rns variant of approximate homomorphic encryption. In *International Conference on Selected Areas in Cryptography*, pp. 347–368. Springer, 2018.

Edward Chou, Josh Beal, Daniel Levy, Serena Yeung, Albert Haque, and Li Fei-Fei. Faster cryptonets: Leveraging sparsity for real-world encrypted inference. *arXiv preprint arXiv:1811.09953*, 2018.

Ronald Cramer, Ivan Bjerre Damgård, and Jesper Buus Nielsen. *Secure Multiparty Computation and Secret Sharing*. Cambridge University Press, 2015.

Anders Dalskov, Daniel Escudero, and Marcel Keller. Fantastic four:{Honest-Majority}{Four-Party} secure computation with malicious security. In *30th USENIX Security Symposium (USENIX Security 21)*, pp. 2183–2200, 2021.

Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure mpc for dishonest majority – or: Breaking the spdz limits. In *18th European Symposium on Research in Computer Security (ESORICS'13)*, pp. 1–18, 2013.

Roshan Dathathri, Olli Saarikivi, Hao Chen, Kim Laine, Kristin Lauter, Saeed Maleki, Madanlal Musuvathi, and Todd Mytkowicz. Chet: An optimizing compiler for fully-homomorphic neural-network inferencing. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI)*, pp. 142–156, 2019.

DESILO. Liberate.FHE: A New FHE Library for Bridging the Gap between Theory and Practice with a Focus on Performance and Accuracy, 2023. `https://github.com/Desilo/liberate-fhe`.

Yvo Desmedt. Threshold cryptosystems. In *Advances in Cryptology (AUSCRYPT '92)*, pp. 1–14, 1993.

Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *IACR Cryptology ePrint Archive*, 2012:144, 2012.

David Froelicher, Hyunghoon Cho, Manaswitha Edupalli, Joao Sa Sousa, Jean-Philippe Bossuat, Apostolos Pyrgelis, Juan R. Troncoso-Pastoriza, Bonnie Berger, and Jean-Pierre Hubaux. Scalable and Privacy-Preserving Federated Principal Component Analysis . In *2023 IEEE Symposium on Security and Privacy (SP'23)*, pp. 1908–1925, 2023.

Craig Gentry. Fully homomorphic encryption using ideal lattices. *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pp. 169–178, 2009.

Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, pp. 201–210, 2016.

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

Intak Hwang, Yisol Hwang, Miran Kim, Dongwon Lee, and Yongsoo Song. Provably secure approximate computation protocols from CKKS. Cryptology ePrint Archive, Paper 2025/395, 2025. URL `https://eprint.iacr.org/2025/395`.

Xiaoqian Jiang, Miran Kim, Kristin Lauter, and Yongsoo Song. Secure outsourced matrix computation and application to neural networks. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pp. 1209–1222, 2018.

Wonkyung Jung, Sangpyo Kim, Jung Ho Ahn, Jung Hee Cheon, and Younho Lee. Over 100x faster bootstrapping in fully homomorphic encryption through memory-centric optimization with gpus. *IACR Transactions on Cryptographic Hardware and Embedded Systems*, pp. 114–148, 2021.

Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. {GAZELLE}: A low latency framework for secure neural network inference. In *27th USENIX security symposium (USENIX security 18)*, pp. 1651–1669, 2018.

Marcel Keller. Mp-spdz: A versatile framework for multi-party computation. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (CCs'20)*, pp. 1575–1590, 2020.

Miran Kim, Xiaoqian Jiang, Kristin Lauter, Elkhan Ismayilzada, and Shayan Shams. Hear: Human action recognition via neural networks on homomorphically encrypted data. *arXiv preprint arXiv:2104.09164*, 2021.

Miran Kim, Xiaoqian Jiang, Kristin Lauter, Elkhan Ismayilzada, and Shayan Shams. Secure human action recognition by encrypted neural network inference. *Nature communications*, 13(1):4799, 2022a.

Sangpyo Kim, Jongmin Kim, Michael Jaemin Kim, Wonkyung Jung, John Kim, Minsoo Rhu, and Jung Ho Ahn. Bts: An accelerator for bootstrappable fully homomorphic encryption. In *Proceedings of the 49th annual international symposium on computer architecture*, pp. 711–725, 2022b.

Nishat Koti, Mahak Pancholi, Arpita Patra, and Ajith Suresh. {SWIFT}: Super-fast and robust {Privacy-Preserving} machine learning. In *30th USENIX Security Symposium (USENIX Security 21)*, pp. 2651–2668, 2021.

Nishant Kumar, Mayank Rathee, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. Cryptflow: Secure tensorflow inference. In *2020 IEEE Symposium on Security and Privacy (SP)*, pp. 336–353. IEEE, 2020.

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

Junghyun Lee, Eunsang Lee, Joon-Woo Lee, Yongjune Kim, Young-Sik Kim, and Jong-Seon No. Precise approximation of convolutional neural networks for homomorphically encrypted data, 2021.

Wentao Li, Miran Kim, Kai Zhang, Han Chen, Xiaoqian Jiang, and Arif Harmanci. Collagene enables privacy-aware federated and collaborative genomic data analysis. *Genome Biology*, 24(1):204, 2023.

Zhengyi Li, Kang Yang, Jin Tan, Wen-jie Lu, Haoqi Wu, Xiao Wang, Yu Yu, Derun Zhao, Yancheng Zheng, Minyi Guo, et al. Nimbus: Secure and efficient two-party inference for transformers. *Advances in Neural Information Processing Systems*, 37:21572–21600, 2024.

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. *Cryptology ePrint Archive*, 2023.

Pratyush Mishra, Ryan Lehmkuhl, Akshayaram Srinivasan, Wenting Zheng, and Raluca Ada Popa. Delphi: a cryptographic inference service for neural networks. In *Proceedings of the 29th USENIX Conference on Security Symposium*, pp. 2505–2522, 2020.

Payman Mohassel and Peter Rindal. Aby3: A mixed protocol framework for machine learning. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, pp. 35–52, 2018.

Payman Mohassel and Yupeng Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (SP)*, pp. 19–38. IEEE, 2017.

Jungho Moon, Dongwoo Yoo, Xiaoqian Jiang, and Miran Kim. Thor: Secure transformer inference with homomorphic encryption. *Cryptology ePrint Archive*, 2024.

Christian Mouchet, Juan Troncoso-Pastoriza, Jean-Philippe Bossuat, and Jean-Pierre Hubaux. Multiparty homomorphic encryption from ring-learning-with-errors. *Proceedings on Privacy Enhancing Technologies*, 2021(4):291–311, 2021.

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

Dongjin Park, Eunsang Lee, and Joon-Woo Lee. Powerformer: Efficient and high-accuracy privacy-preserving language model with homomorphic encryption. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11090–11111, 2025a.

Junyoung Park, Dalton Jones, Matthew J Morse, Raghavv Goel, Mingu Lee, and Chris Lott. Keydiff: Key similarity-based kv cache eviction for long-context llm inference in resource-constrained environments. *arXiv preprint arXiv:2504.15364*, 2025b.

Hongwu Peng, Ran Ran, Yukui Luo, Jiahui Zhao, Shaoyi Huang, Kiran Thorat, Tong Geng, Chenghong Wang, Xiaolin Xu, Wujie Wen, et al. Lingcn: Structural linearized graph convolutional network for homomorphically encrypted inference. *Advances in Neural Information Processing Systems*, 36:20679–20694, 2023.

Ran Ran, Wei Wang, Quan Gang, Jieming Yin, Nuo Xu, and Wujie Wen. Cryptogcn: Fast and scalable homomorphically encrypted graph convolutional network inference. *Advances in Neural Information Processing Systems*, 35:37676–37689, 2022.

Ran Ran, Xinwei Luo, Wei Wang, Tao Liu, Gang Quan, Xiaolin Xu, Caiwen Ding, and Wujie Wen. SpENCNN: Orchestrating encoding and sparsity for fast homomorphically encrypted neural network inference. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202, pp. 28718–28728. PMLR, 2023.

Nikola Samardzic, Axel Feldmann, Aleksandar Krastev, Nathan Manohar, Nicholas Genise, Srinivas Devadas, Karim Eldefrawy, Chris Peikert, and Daniel Sanchez. Craterlake: a hardware accelerator for efficient unbounded computation on encrypted data. In *ISCA*, pp. 173–187, 2022.

Nigel P Smart and Frederik Vercauteren. Fully homomorphic encryption with relatively small key and ciphertext sizes. In *International Workshop on Public Key Cryptography*, pp. 420–443. Springer, 2010.

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

Jianming Tong, Jingtian Dang, Anupam Golder, Arijit Raychowdhury, Cong Hao, and Tushar Krishna. Accurate low-degree polynomial approximation of non-polynomial operators for fast private inference in homomorphic encryption. *Proceedings of Machine Learning and Systems*, 6: 210–223, 2024.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

U.S. Congress. Health insurance portability and accountability act of 1996. Public Law 104–191, 1996. URL `https://www.congress.gov/bill/104th-congress/house-bill/3103`.

Xi Xie, Ran Ran, Jiahui Zhao, Bin Lei, Zhijie Shi, Wujie Wen, and Caiwen Ding. Uld-net: Enabling ultra-low-degree fully polynomial networks for homomorphically encrypted inference. In *International Conference on Learning Representations*, volume 2026, pp. 9406–9419, 2026.

Tianshi Xu, Wen-jie Lu, Jiangrui Yu, Yi Chen, Chenqi Lin, Runsheng Wang, and Meng Li. Breaking the layer barrier: Remodeling private transformer inference with hybrid {CKKS} and {MPC}. In *34th USENIX Security Symposium (USENIX Security 25)*, pp. 2653–2672, 2025.

Ye Yu, Yifan Zhou, Yi Chen, Pedro Soto, Wenjie Xiong, and Meng Li. Cachemir: Fully homomorphic encrypted inference of generative large language model with kv cache. *arXiv preprint arXiv:2602.11470*, 2026.

Wenxuan Zeng, Ye Dong, Jinjin Zhou, Jin Tan, Lei Wang, Tao Wei, Runsheng Wang, and Meng Li. Mpcache: Mpc-friendly kv cache eviction for efficient private llm inference. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025.

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 *Network and Distributed System Security Symposium (NDSS)*, 2025. doi: 10.14722/ndss.2025.230868. URL `https://www.ndss-symposium.org/ndss-paper/secure-transformer-inference-made-non-interactive/`.

A NOTATION

Tables 4–7 summarize the notation used throughout the paper and its appendices, grouped by topic.

Table 4: System and Multi-Party Architecture Notation

Symbol	Definition
N_S	Number of servers in the collaborative cluster (Sec. 3.1)
$1 \le N_{\text{comp}} \le N_S$	Number of servers authorized to act as computation servers in a deployment
S_i	Compute server assigned to the current client session c
S_j	A peer (helper/threshold) server, $j \in \mathcal{T}_i$
$\mathcal{T}_i$	Peer-server index set for the session assigned to S_i, $\mathcal{T}_i = \{1, \dots, N_S\} \setminus \{i\}$
n	Number of threshold (peer) parties, $n = N_S - 1$ (Appendix D)
N_C	Maximum active client sessions per computation server
$N_{C,i}$	Maximum concurrent clients server S_i can serve (Appendix E)
c	Client session identifier
sid_c	Unique session identifier generated for session c (Appendix C)
$\mathcal{A}$	Adversary coalition of corrupted servers, $\lvert\mathcal{A}\rvert \le N_S - 2$ (Appendix L)

Table 5: CKKS / Homomorphic Encryption Notation

Symbol	Definition
ℓ_{ct}	Current (multiplicative) noise level of a ciphertext
$\ell_{\max}$	Maximum (refreshed) noise level of a ciphertext
Q	Ciphertext modulus
D	Ring/polynomial degree (CKKS parameter)
R_Q	Polynomial ring $\mathbb{Z}_Q[X]/(X^D + 1)$
pk, sk	Public key, secret key (generic, session-independent form)
$pk^{(c)}$, $sk^{(c)}$	Public key, secret key for client session c
$sk_j^{(c)}$	Secret-key share for session c held by peer S_j, $sk^{(c)} = \sum_{j \in \mathcal{T}_i} sk_j^{(c)} \bmod Q$
$\mathbf{evk}_{\text{mult}}$	Evaluation (relinearization) key for $CMult$
$\mathbf{evk}_{\text{rot}}^{(\rho)}$	Evaluation key for Rot with rotation amount ρ
$\mathbf{evk}^{(c)}$	Evaluation keys for session c: $\mathbf{evk}_{\text{mult}}^{(c)}$ and $(\mathbf{evk}_{\text{rot}}^{(\rho)})^{(c)}$
λ	Security level, in bits
Δ	CKKS scale factor (Appendix H.1–H.2)
$ct = (c_0, c_1)$	Ciphertext, as a pair of ring elements
$\text{Enc}_{pk}(\cdot)$, $\text{Dec}_{sk}(\cdot)$	Encryption; full decryption
$\text{PDec}_{sk_j}(\cdot)$	Partial decryption under share sk_j
$e_{sm,j}$	Smudging noise added by peer S_j during partial decryption (Eq. equation 1)
M	Aggregate mask (additive or multiplicative), $M = \sum_{j \in \mathcal{T}_i} M_j$
M_j	Mask share held by peer S_j

Table 6: Protocol and Sub-Protocol Notation

Symbol	Definition
$\mathbf{P}_{\text{refresh}}$	Noise refresh protocol (Alg. 2, Sec. 4.1)
$\mathbf{P}_{\text{nonlinear}}$	Masked non-linear evaluation protocol; orchestrates $\mathbf{P}_{\text{MSoftmax}}$/$\mathbf{P}_{\text{MLayerNorm}}$/$\mathbf{P}_{\text{MSiLU}}$ (Alg. 3, Sec. 4.2)
$\mathbf{P}_{\text{KVM}}$	Masked KV-cache update protocol (Alg. 4, Sec. 4.3)
$\mathbf{P}_{\text{unmaskmul}}$	Ciphertext–masked-plaintext product with mask cancellation (Alg. 5, Sec. 4.3)
$\mathbf{P}_{\text{exp}}$	Masked exponential primitive, $\phi = \exp(\cdot)$ (Alg. 6)
$\mathbf{P}_{\text{inv}}$	Masked reciprocal primitive, $\phi = 1/(\cdot)$ (Alg. 7)
$\mathbf{P}_{\text{invsqrt}}$	Masked inverse-square-root primitive, $\phi = 1/\sqrt{\cdot}$ (Alg. 8)
$\mathbf{P}_{\text{MSoftmax}}$	Masked Softmax evaluation, composes $\mathbf{P}_{\text{exp}}$ and $\mathbf{P}_{\text{inv}}$ (Alg. 9)
$\mathbf{P}_{\text{MLayerNorm}}$	Masked LayerNorm evaluation, uses $\mathbf{P}_{\text{invsqrt}}$ (Alg. 10)
$\mathbf{P}_{\text{MSiLU}}$	Masked SiLU evaluation, composes $\mathbf{P}_{\text{exp}}$ and $\mathbf{P}_{\text{inv}}$ (Alg. 11)
ϕ	Target non-linear function inside a masked-evaluation primitive ($\exp$, $1/(\cdot)$, $1/\sqrt{\cdot}$)
g	Mask-correction function associated with ϕ inside $\mathbf{P}_{\text{nonlinear}}$

Table 7: LLM, Transformer, and Circuit Notation

Symbol	Definition
L	Number of transformer layers
$\mathcal{C} = \{\mathcal{C}_l\}_{l=1}^{L}$	Layered computation circuit for the transformer; $\mathcal{C}_l$ = circuit operations in layer l
op_c	A single circuit operation (a node in $\mathcal{C}_l$)
$Q_t^{(l)}, K_t^{(l)}, V_t^{(l)}$	Query, key, value vectors at decoding step t, layer l
d_k	Key dimension
$H_t^{(l)}$	Attention logits at step t, layer l, $H_t^{(l)} = \langle Q_t^{(l)}, K_{1:t}^{(l)} \rangle$
$A_t^{(l)}$	Attention weights, $A_t^{(l)} = \mathrm{Softmax}(H_t^{(l)})$
$O_t^{(l)}$	Attention output at step t, layer l, $O_t^{(l)} = \langle A_t^{(l)}, V_{1:t}^{(l)} \rangle$
$\boldsymbol{\gamma}, \boldsymbol{\beta}$	LayerNorm affine scale and shift parameters
ϵ	LayerNorm numerical-stability constant
$\sigma(\cdot)$	Sigmoid function, used in the SiLU definition

B Additional Details on CKKS

CKKS allows arithmetic operations on encrypted data over fixed-point numbers with predefined precision, which makes it an ideal candidate for performing machine learning tasks where most of the computations are approximate. The supported homomorphic operations include ciphertext addition *Add* $\sim (ct_1 + ct_2)$, plaintext – ciphertext multiplication *PMult* $\sim (ct \times pt)$, ciphertext – ciphertext multiplication *CMult* $\sim (ct_1 \times ct_2)$ which requires the relinearization key $\mathbf{evk}_{\text{mult}}$ for key-switching, and ciphertext rotation *Rot* $\sim (ct, k)$ which requires the rotation key $\mathbf{evk}_{\text{rot}}^{(\rho)}$ for key-switching.

In CKKS, *Rot*, together with *CMult*, incurs much higher overhead than other HE operations, as it involves complex automorphism and key-switching steps Samardzic et al. (2022).

Compared with unencrypted operations, CKKS operations incur substantial runtime and memory overhead. Packing enables multiple data values to be stored in a single ciphertext, allowing computations to be performed in a Single Instruction Multiple Data (SIMD) fashion Smart & Vercauteren (2010; 2014). In this work, we leverage this property to reduce amortized overhead and maximize SIMD processing efficiency for both inference and fine-tuning, applying it to initial input packing as well as intermediate ciphertexts throughout the ML processing pipeline.

For a leveled HE, the maximum level of a ciphertext ($\ell_{\max}$) is defined by the number of consecutive multiplications that can be performed without bootstrapping. Each homomorphic multiplication decreases the level by one, necessitating a rescale operation. Once the level reaches zero, bootstrapping is required to refresh the ciphertext and enable further computation. With appropriate encryption parameters (N, Q), the scheme achieves a security level of $\lambda = 128$ bits Chase et al. (2017), meaning that breaking the encryption would require approximately 2^{128} operations.

While this work primarily targets CKKS, our techniques can be adapted to other leveled HE schemes such as BGV Brakerski et al. (2014) and BFV Fan & Vercauteren (2012), which share similar arithmetic operations.

C Key Generation and Distribution

We summarize the key generation and distribution protocol as follows:

Session Initialization and Key Generation. When a client is assigned to S_i for inference processing, the system then generates a unique session identifier sid_c for this client's session. Furthermore, the trusted key generator, upon receiving the session request with parameters (security level λ, number of servers N_S, sid_c, and S_i), produces $\{pk^{(c)}, \mathbf{evk}^{(c)}, sk^{(c)}\}$ specifically for this client session. Here $\mathbf{evk}^{(c)}$ denotes the set of evaluation keys, comprising both relinearization key $\mathbf{evk}_{\text{mult}}^{(c)}$ and rotation key $(\mathbf{evk}_{\text{rot}}^{(\rho)})^{(c)}$.

Secret Key Splitting and Key Distribution. To enable threshold-based noise refresh while preserving client's ability to decrypt final results, the secret key $sk^{(c)}$ must be distributed in two forms:

as a complete key to the client, and as additive shares to peer servers $\{sk_j^{(c)}\}_{j=1,j\neq i}^{N_S}$. Specifically, the key generator samples $N_S - 2$ shares independently and uniformly at random from the key space, and computes the final share such that

$$sk^{(c)} = \sum_{j \in T_j} sk_j^{(c)} \bmod Q \ ,$$

where $\mathcal{T}_i = \{1, \ldots, N_S\} \setminus \{i\}$. Furthermore, the key generator distributes the cryptographic material over authenticated and encrypted channels to *three* distinct recipients:

- The client receives sid_c, assigned server identity i, $pk^{(c)}$, and the full secret key $sk^{(c)}$ which enables the client to decrypt inference results
- S_i receives sid_c, $pk^{(c)}$, and $\mathbf{evk}^{(c)}$ for performing homomorphic computations on the client's encrypted queries.
- Each peer server S_j where $j \in \mathcal{T}_i$ receives sid_c, i, and its designated secret key share $sk_j^{(c)}$. These shares enable peer servers to conduct threshold noise refresh operations during the client's session.

Cleanup. After successful distribution of all key material, the trusted key generator securely deletes the complete secret key $sk^{(c)}$, all intermediate shares, and any other sensitive material from its memory and storage.

Distributed Key Generation. The trusted key generator above is a deployment convenience, not a requirement of the protocol itself: *SHIM* also supports replacing it with a standard Distributed Key Generation (DKG) protocol Mouchet et al. (2021) run by S_i and the peers, so that no single party, including the key generator, ever observes $sk^{(c)}$ in the clear. We discuss this instantiation in Remark L.7 and refer the reader there for details.

D Secret Sharing and Threshold Cryptography

We employ the common *additive secret sharing* Cramer et al. (2015); Damgård et al. (2013), also known as *linear secret sharing*, as the foundation for enabling collaborative and privacy-preserving decryption in our multi-server homomorphic encryption framework. Specifically, a secret key $sk \in \mathbb{F}$ ($\mathbb{F}$ is a finite field) is split into n shares where typically the first $n-1$ shares are sampled uniformly at random: $sk_1, sk_2, \ldots, sk_{n-1} \xleftarrow{\$} \mathbb{F}$, and the final share is computed as $sk_n = sk - \sum_{i=1}^{n-1} sk_i$. Each party receives a secret key share sk_i. During threshold decryption, given a ciphertext $ct = \text{Enc}_{pk}(X)$ where X is the plaintext, each party computes a partial decryption share by applying sk_i on the ciphertext ct as $d_i = \text{PDec}_{sk_i}(ct)$ where $\text{Enc}_{pk}(\cdot)$ and $\text{PDec}_{sk_i}(\cdot)$ are the encryption (under public key pk) and partial decryption (under sk_i) operations, respectively. These shares are then combined to reconstruct the full decryption output: $\text{Dec}_{sk}(ct) = X = \sum_{i \in T} d_i$ with $sk = \sum_{i \in T} sk_i$ where $\text{Dec}_{sk}(\cdot)$ is the decryption operation with sk, and $T = \{1, \ldots, n\}$. This approach ensures that no single party ever holds the full secret key, and the ciphertext can only be decrypted if all parties cooperate, thus achieving $(n-1)$-privacy: any coalition of fewer than n servers (up to $n-1$ corrupted parties) learns no information about the plaintext or secret key. Mapping to our setting, now we have $n = N_S - 1$.

Note as in CKKS-based systems, secret keys live in the polynomial ring $R_Q = \mathbb{Z}_Q[X]/(X^D + 1)$, where Q is the ciphertext modulus and D is the polynomial degree. Hence we apply additive secret sharing *coefficient-wise* to the polynomial representation. More specifically, we can treat an element $sk \in R_Q$ as a vector of D coefficients in $\mathbb{Z}_Q$, apply standard additive secret sharing independently to each coefficient, and reconstruct shares that are themselves elements of R_Q.

E Multi-Client Support

SHIM also improves multi-client serving by reducing the per-client memory footprint during decoding. In conventional FHE-based serving, each active session must maintain its own encrypted KV cache.

Let $b_{\mathrm{KV}}^{\mathrm{ct}}$ denote the memory footprint of one ciphertext KV cache and $b_{\mathrm{KV}}^{\mathrm{mask}}$ denote the footprint of SHIM's masked-plaintext KV cache. Since $b_{\mathrm{KV}}^{\mathrm{mask}} \ll b_{\mathrm{KV}}^{\mathrm{ct}}$, the number of resident client sessions allowed by the memory budget of server S_i increases from

$$\left\lfloor \frac{B_i}{b_{\mathrm{KV}}^{\mathrm{ct}}} \right\rfloor \quad \text{to} \quad \left\lfloor \frac{B_i}{b_{\mathrm{KV}}^{\mathrm{mask}}} \right\rfloor \ , \tag{5}$$

where B_i is the memory budget reserved for active session states. The corresponding masks and key shares remain distributed across peer servers, so co-locating more masked KV caches on S_i does not expose the cached states.

The serving parallelism of a primary server is nevertheless not determined by memory alone. Let $|\mathcal{C}_i|$ denote the number of computation nodes at S_i, and let n_{dec} denote the number of computation nodes required by one active decoding stream. Then the number of clients that S_i can process concurrently is bounded by

$$N_{C,i} \le \min\left(\left\lfloor \frac{B_i}{b_{\mathrm{KV}}^{\mathrm{mask}}} \right\rfloor , \left\lfloor \frac{|\mathcal{C}_i|}{n_{\mathrm{dec}}} \right\rfloor \right) \ . \tag{6}$$

Across the cluster, the scheduler routes clients according to both terms in this bound, yielding an aggregate capacity that scales with $\sum_i N_{C,i}$ rather than memory alone. Importantly, this multi-client support does not weaken the privacy guarantee: each client session uses independent keys and masks, and the primary computation server never holds the secret-key shares for the client it serves.

F KV Cache Management and Computation Flow

Autoregressive decoding and attention notation. Given an input query q, an autoregressive LLM $\mathcal{M}$ assigns probability to an output sequence $\mathbf{a}_{1:n_L}$ as

$$P_{\mathcal{M}}(\mathbf{a}_{1:n_L} \mid q) = \prod_{t=1}^{n_L} P_{\mathcal{M}}(a_t \mid \mathbf{a}_{<t}, q), \qquad \mathbf{a}_{<t} = \mathbf{a}_{1:t-1}. \tag{7}$$

At decoding step t and layer l, attention uses the current query $Q_t^{(l)}$ and the key/value vectors cached through step t:

$$\mathrm{Attn}_t^{(l)} = \mathrm{Softmax}\left(\frac{Q_t^{(l)} (K_{1:t}^{(l)})^{\mathsf{T}}}{\sqrt{d_k}} \right) V_{1:t}^{(l)}, \tag{8}$$

where d_k is the key dimension. The cache stores the previously computed key/value pairs at each layer; at the next decoding step, only the newly computed pair is appended. The masked cache update and the subsequent attention products are detailed below. In generative LLMs, the KV cache grows linearly with the decoding length at every layer, which creates significant memory and throughput pressure. This issue is further exacerbated in privacy preserving settings when the KV cache is stored under CKKS, since ciphertexts have a much larger footprint than plaintexts. To control memory usage and sustain decoding throughput, our framework adopts a scheduling strategy that decouples KV cache maintenance from the main inference pipeline. Concretely, we offload KV cache operations to a dedicated KV management process: while one process executes the forward pass and attention logic, another process securely manages KV storage and masking.

Prefill stage. At layer l, after applying the key and value projection matrices $W_K^{(l)}$ and $W_V^{(l)}$ to the hidden states, the server produces encrypted cache matrices $(\mathrm{Enc}_{pk}(K_1^{(l)}), \mathrm{Enc}_{pk}(V_1^{(l)}))$. The KV management process in S_i sends them to the threshold parties. Each threshold party $j \in \mathcal{T}_i$ performs partial decryption and adds its local masks $M_{K,1}^{(l),j}$ and $M_{V,1}^{(l),j}$ which are independently and respectively generated by following the distribution of M (Sec. H.1) scaled by $1/(N_S - 1)$. Here the superscript 1 denotes caches generated from the *input sequence*. Concretely, the index 1 marks the first entry in the same running index later used for t in the decoding stage: unlike each subsequent decoding step $t = 2, 3, \ldots$, which adds exactly one new token's key/value pair, this first entry is produced by the prefill pass and therefore represents the entire input sequence's worth of cache computed in parallel, not a single token position.

Each party returns only the masked partial decryption shares to S_i and keeps the masks locally. After aggregation, S_i stores the masked plaintext caches $K_1^{(l)} + M_{K,1}^{(l)}$ and $V_1^{(l)} + M_{V,1}^{(l)}$ with $M_{K,1}^{(l)} = \sum_{j\in\mathcal{T}_i} M_{K,1}^{(l),j}$ and $M_{V,1}^{(l)} = \sum_{j\in\mathcal{T}_i} M_{V,1}^{(l),j}$.

Auto regressive decoding stage. At decoding step t in layer l, the hidden state is projected to obtain encrypted query, key, and value vectors: $\text{Enc}_{pk}(Q_t^{(l)})$, $\text{Enc}_{pk}(K_t^{(l)})$, and $\text{Enc}_{pk}(V_t^{(l)})$. The KV management process applies the same procedure to $\text{Enc}_{pk}(K_t^{(l)})$ and $\text{Enc}_{pk}(V_t^{(l)})$, yielding masked plaintext vectors $K_t^{(l)} + M_{K,t}^{(l)}$ and $V_t^{(l)} + M_{V,t}^{(l)}$, where $M_{K,t}^{(l)} = \sum_{j\in\mathcal{T}_i} M_{K,t}^{(l),j}$ and $M_{V,t}^{(l)} = \sum_{j\in\mathcal{T}_i} M_{V,t}^{(l),j}$. We append them to the existing caches:

$$\begin{aligned} K_{1:t}^{(l)} + M_{K,1:t}^{(l)} &= \big[\, K_{1:t-1}^{(l)} + M_{K,1:t-1}^{(l)} \;:\; K_t^{(l)} + M_{K,t}^{(l)} \,\big], \\ V_{1:t}^{(l)} + M_{V,1:t}^{(l)} &= \big[\, V_{1:t-1}^{(l)} + M_{V,1:t-1}^{(l)} \;:\; V_t^{(l)} + M_{V,t}^{(l)} \,\big] \;. \end{aligned} \tag{9}$$

The prefill and auto-regressive cache-update steps described above are formalized as the $\mathbf{P}_{\text{KVM}}$ protocol, given as Alg. 4 in Appendix G.

Parallel mask cancellation for attention logits computation. To compute attention logits at step t in layer l, we need $H_t^{(l)} = \langle Q_t^{(l)},\, K_{1:t}^{(l)} \rangle$, where $\langle\cdot,\cdot\rangle$ stands for the inner product. Since S_i stores only the masked key cache $K_{1:t}^{(l)} + M_{K,1:t}^{(l)}$, it first computes masked logits in the HE domain:

$$\begin{aligned} \text{Enc}_{pk}\Big(H_{\text{mask},t}^{(l)}\Big) &= \Big\langle \text{Enc}_{pk^{(c)}}\Big(Q_t^{(l)}\Big),\; K_{1:t}^{(l)} + M_{K,1:t}^{(l)} \Big\rangle \\ &= \text{Enc}_{pk}\Big(H_t^{(l)} + \Big\langle Q_t^{(l)},\, M_{K,1:t}^{(l)} \Big\rangle\Big) \;. \end{aligned} \tag{10}$$

We cancel the mask term in parallel across threshold parties without revealing $Q_t^{(l)}$ in plaintext. S_i first sends $\text{Enc}_{pk^{(c)}}(Q_t^{(l)})$ to the peers $\{S_j\}_{j\in\mathcal{T}_i}$. Each party $j \in \mathcal{T}_i$ holds its plaintext mask share $M_{K,1:t}^{(l),j}$ and computes the encrypted contribution via a ciphertext-plaintext inner product where we denote $h_{K,t}^{(l),j} = \langle Q_t^{(l)}, M_{K,1:t}^{(l),j} \rangle$:

$$\text{Enc}_{pk}\Big(h_{K,t}^{(l),j}\Big) \triangleq \Big\langle \text{Enc}_{pk}\Big(Q_t^{(l)}\Big),\; M_{K,1:t}^{(l),j} \Big\rangle \;. \tag{11}$$

Each party sends only $\text{Enc}_{pk}(h_{K,t}^{(l),j})$ to S_i, and S_i aggregates

$$\text{Enc}_{pk}\Big(h_{K,t}^{(l)}\Big) \triangleq \sum_{j\in\mathcal{T}_i} \text{Enc}_{pk}\Big(h_{K,t}^{(l),j}\Big) = \text{Enc}_{pk}\Big(\Big\langle Q_t^{(l)},\, M_{K,1:t}^{(l)} \Big\rangle\Big) \;, \tag{12}$$

where $M_{K,1:t}^{(l)} = \sum_{j\in\mathcal{T}_i} M_{K,1:t}^{(l),j}$. Finally, S_i removes the mask term by homomorphic subtraction:

$$\text{Enc}_{pk}\Big(H_t^{(l)}\Big) = \text{Enc}_{pk}\Big(H_{\text{mask},t}^{(l)}\Big) - \text{Enc}_{pk}\Big(h_{K,t}^{(l)}\Big) \;. \tag{13}$$

This design parallelizes the mask related computation across threshold parties and avoids encrypting or transmitting masks. The only messages returned are encrypted intermediate contributions, which are aggregated by S_i.The protocol for a ciphertext–masked-plaintext product with mask cancellation, as derived above, is summarized as $\mathbf{P}_{\text{unmaskmul}}$ (Algorithm 5). The derivation above (from Eq. (10) to Eq. (13)) is exactly this protocol applied with $ct = \text{Enc}_{pk^{(c)}}(X) = \text{Enc}_{pk^{(c)}}(Q_t^{(l)})$ and $Y_{op_c} + M_{Y_{op_c}} = K_{1:t}^{(l)} + M_{K,1:t}^{(l)}$, recovering $\text{Enc}_{pk^{(c)}}(H_t^{(l)})$.

Parallel mask cancellation for the attention output computation. After $\mathbf{P}_{\text{nonlinear}}$ (Alg. 3) evaluates $A_t^{(l)} = \text{Softmax}(H_t^{(l)})$ under encryption, to compute the attention output at step t in layer l, we need $O_t^{(l)} = \langle A_t^{(l)}, V_{1:t}^{(l)} \rangle$. Similarly, since S_i stores only the masked value cache $V_{1:t}^{(l)} + M_{V,1:t}^{(l)}$, it first computes the masked output in the HE domain:

$$\text{Enc}_{pk^{(c)}}\Big(O_{\text{mask},t}^{(l)}\Big) = \Big\langle \text{Enc}_{pk^{(c)}}\Big(A_t^{(l)}\Big),\; V_{1:t}^{(l)} + M_{V,1:t}^{(l)} \Big\rangle = \text{Enc}_{pk^{(c)}}\Big(O_t^{(l)} + \Big\langle A_t^{(l)}, M_{V,1:t}^{(l)} \Big\rangle\Big) \;. \tag{14}$$

Algorithm 2 $\mathbf{P}_{\text{refresh}}$: Noise Refresh Protocol (Sec. 4.1)

Input: Ciphertext $ct = (c_0, c_1) = \text{Enc}_{pk^{(c)}}(X)$ with exhausted noise level $\ell_{ct} = 0$

S_i sends ct to peers $\{S_j\}_{j \in \mathcal{T}_i}$
for each $j \in \mathcal{T}_i$, in parallel, S_j **do**
  Compute partial decryption $d_j = \text{PDec}_{sk_j^{(c)}}(ct) = c_1 \cdot sk_j^{(c)} + e_{sm,j} \bmod Q$ (Eq. 1)
  Sample additive mask share M_j from the distribution of M (Sec. H.1) scaled by $1/(N_S - 1)$
  Compute masked partial decryption $d_j + M_j$
  Send $\big(d_j + M_j,\ \text{Enc}_{pk^{(c)}}(M_j)\big)$ to S_i
end for
S_i collects $N_S - 1$ masked partial decryptions and $N_S - 1$ encrypted mask terms
S_i reconstructs the masked plaintext: $X + M = c_0 + \sum_{j \in \mathcal{T}_i}(d_j + M_j) \mod Q$ (Eq. 2)
S_i aggregates the encrypted mask: $\text{Enc}_{pk^{(c)}}(M) = \sum_{j \in \mathcal{T}_i} \text{Enc}_{pk^{(c)}}(M_j)$
S_i removes the encrypted mask: $\text{Enc}_{pk^{(c)}}(X) = (X + M) - \text{Enc}_{pk^{(c)}}(M)$
Refreshing the ciphertext: $\ell_{ct} \leftarrow \ell_{max}$

Output: Refreshed ciphertext $\text{Enc}_{pk^{(c)}}(X)$ with noise level $\ell_{ct} = \ell_{max}$

We cancel the mask term in parallel across threshold parties without revealing $A_t^{(l)}$ in plaintext. Same as the steps in mask cancellation for atention logits, at beginning, S_i sends $\text{Enc}_{pk^{(c)}}(A_t^{(l)})$ to the peers $\{S_j\}_{j \in \mathcal{T}_i}$, and each party $j \in \mathcal{T}_i$ who holds its plaintext mask share $M_{V,1:t}^{(l),j}$ computes the encrypted contribution via a ciphertext-plaintext inner product:

$$\text{Enc}_{pk^{(c)}}\left(o_{V,t}^{(l),j}\right) \triangleq \left\langle \text{Enc}_{pk^{(c)}}\left(A_t^{(l)}\right),\ M_{V,1:t}^{(l),j} \right\rangle \quad , \tag{15}$$

here $o_{V,t}^{(l),j} = \langle A_t^{(l)}, M_{V,1:t}^{(l),j} \rangle$. Consequently, each party sends only $\text{Enc}_{pk^{(c)}}\left(o_{V,t}^{(l),j}\right)$ to S_i, and S_i aggregates

$$\text{Enc}_{pk^{(c)}}\left(o_{V,t}^{(l)}\right) \triangleq \sum_{j \in \mathcal{T}_i} \text{Enc}_{pk^{(c)}}\left(o_{V,t}^{(l),j}\right) = \text{Enc}_{pk^{(c)}}\left(\left\langle A_t^{(l)}, M_{V,1:t}^{(l)} \right\rangle\right) \quad , \tag{16}$$

where $M_{V,1:t}^{(l)} = \sum_{j \in \mathcal{T}_i} M_{V,1:t}^{(l),j}$. Finally, S_i removes the mask term by homomorphic subtraction:

$$\text{Enc}_{pk^{(c)}}\left(O_t^{(l)}\right) = \text{Enc}_{pk^{(c)}}\left(O_{\text{mask},t}^{(l)}\right) - \text{Enc}_{pk^{(c)}}\left(o_{V,t}^{(l)}\right) \quad . \tag{17}$$

Same as the mask cancellation for attention logits, the only messages returned are encrypted intermediate contributions, which are aggregated by S_i. The derivation above (from Eq. (14) to Eq. (17)) is exactly protocol $\mathbf{P}_{\text{unmaskmul}}$ (Algorithm 5) applied with $ct = \text{Enc}_{pk^{(c)}}(X) = \text{Enc}_{pk^{(c)}}(A_t^{(l)})$ and $Y_{op_c} + M_{Y_{op_c}} = V_{1:t}^{(l)} + M_{V,1:t}^{(l)}$ to recover $\text{Enc}_{pk^{(c)}}(O_t^{(l)})$.

G Detailed Algorithms for SHIM Sub-Protocols

Alg. 1 invokes three sub-protocols during inference: $\mathbf{P}_{\text{refresh}}$ (Sec. 4.1), triggered when the ciphertext's noise budget is exhausted; the masked non-linear evaluation protocol $\mathbf{P}_{\text{nonlinear}}$ (Sec. 4.2), triggered before non-linear operations such as Softmax, LayerNorm, and SiLU; the KV-cache management protocol $\mathbf{P}_{\text{KVM}}$ (Sec. 4.3 and Sec. F), triggered when new KV pairs are generated during decoding; and the ciphertext–masked-plaintext product with mask-cancellation protocol $\mathbf{P}_{\text{unmaskmul}}$ (Sec. 4.3 and Sec. F), invoked twice per attention block: once to compute attention logits against the masked key cache, and once to compute the attention output against the masked value cache. We give the formal algorithmic description of each below.

Algorithm 3 $\mathbf{P}_{\text{nonlinear}}$: Masked Non-Linear Evaluation Protocol (Sec. 4.2)

Input: Ciphertext $ct = \text{Enc}_{pk^{(c)}}(X)$, circuit operation $op_c \in \{\text{Softmax}, \text{LayerNorm}, \text{SiLU}\}$

If op_c = Softmax
  Call mased Siftmax evaluation protocol $\mathbf{P}_{\text{MSoftmax}}$ (Alg. 9): $ct' \leftarrow \mathbf{P}_{\text{MSoftmax}}(ct)$
Else if op_c = LayerNorm
  Call masked LayerNorm evaluation protocol $\mathbf{P}_{\text{MLayerNorm}}$ (Alg. 10): $ct' \leftarrow \mathbf{P}_{\text{MLayerNorm}}(ct)$
Else (opc = SiLU)
  Call masked SiLU evaluation protocol $\mathbf{P}_{\text{MSiLU}}$ (Alg. 11): $ct' \leftarrow \mathbf{P}_{\text{MSiLU}}(ct)$
End
$\text{Enc}_{pk^{(c)}}(op_c(X)) = ct'$

Output: $\text{Enc}_{pk^{(c)}}(op_c(X))$

Algorithm 4 $\mathbf{P}_{\text{KVM}}$: Masked KV Cache Update Protocol (Sec. 4.3 and Appendix F)

Input: Newly generated encrypted key/value vectors $\left(\text{Enc}_{pk^{(c)}}(K_t^{(l)}),\ \text{Enc}_{pk^{(c)}}(V_t^{(l)})\right)$ at layer l, current step t, existing masked cache $\left(K_{1:t-1}^{(l)}+M_{K,1:t-1}^{(l)},\ V_{1:t-1}^{(l)}+M_{V,1:t-1}^{(l)}\right)$

Prefilling: When $t = 1$, $\left(\text{Enc}_{pk^{(c)}}(K_1^{(l)}), \text{Enc}_{pk^{(c)}}(V_1^{(l)})\right)$ are the the full-prompt matrices
S_i sends $\left(\text{Enc}_{pk^{(c)}}(K_t^{(l)}),\ \text{Enc}_{pk^{(c)}}(V_t^{(l)})\right)$ to peers $\{S_j\}_{j\in\mathcal{T}_i}$
for each $j \in \mathcal{T}_i$, in parallel, S_j **do**
  Compute partial decryptions of $\left(\text{Enc}_{pk^{(c)}}(K_t^{(l)}),\ \text{Enc}_{pk^{(c)}}(V_t^{(l)})\right)$, respectively:
  Sample local mask shares $M_{K,t}^{(l),j}, M_{V,t}^{(l),j}$ from the distribution of M (Sec. H.1) scaled by $1/(N_S-1)$, respectively
  Compute masked partial decryption:
    $\left(\text{PDec}_{sk_j^{(c)}}(\text{Enc}_{pk^{(c)}}(K_t^{(l)})) + M_{K,t}^{(l),j},\ \text{PDec}_{sk_j^{(c)}}(\text{Enc}_{pk^{(c)}}(V_t^{(l)})) + M_{V,t}^{(l),j}\right)$
  Send masked partial decryptions to S_i
end for
S_i aggregates to obtain $\left(K_t^{(l)}+M_{K,t}^{(l)},\ V_t^{(l)}+M_{V,t}^{(l)}\right)$, with $M_{K,t}^{(l)} = \sum_j M_{K,t}^{(l),j}$ and $M_{V,t}^{(l)} = \sum_j M_{V,t}^{(l),j}$
S_i appends the results to cache (Eq. (9)):
    $K_{1:t}^{(l)}+M_{K,1:t}^{(l)} \leftarrow [K_{1:t-1}^{(l)}+M_{K,1:t-1}^{(l)} : K_t^{(l)}+M_{K,t}^{(l)}]$
    $V_{1:t}^{(l)}+M_{V,1:t}^{(l)} \leftarrow [V_{1:t-1}^{(l)}+M_{V,1:t-1}^{(l)} : V_t^{(l)}+M_{V,t}^{(l)}]$

Output: Updated masked cache $\left(K_{1:t}^{(l)}+M_{K,1:t}^{(l)},\ V_{1:t}^{(l)}+M_{V,1:t}^{(l)}\right)$

H Mask Generation Protocols

H.1 Additive Masking at Scale

Let $x \in \mathbb{R}$ denote the (decoded) plaintext value, represented in CKKS at scale Δ. To hide x after decryption while preserving the Δ-grid granularity, we sample an additive mask on the same scale:

$$m \triangleq \Delta x \in \mathbb{Z}\ , \qquad \text{and} \qquad \tilde{m} \triangleq m + r\ ,$$

with

$$r \xleftarrow{\$} \{-R, -R+1, \ldots, R\}\ .$$

The parameter R controls the amount of discrete randomization; as with the multiplicative mask below, the bound we derive from it is an error bound, not a privacy guarantee on its own (see Remark L.6 in Appendix L for the privacy-side implication of this same quantity). Equivalently, in real form the revealed (decrypted) masked value is

$$\tilde{x} \triangleq x + \frac{r}{\Delta}\ .$$

Algorithm 5 $\mathbf{P}_{\text{unmaskmul}}$: Ciphertext–Masked-Plaintext Product Protocol (Sec. 4.3 and Appendix F)

Input: Ciphertext $ct = \text{Enc}_{pk^{(c)}}(X)$, masked plaintext operand $Y_{op_c}+M_{Y_{op_c}}$ with mask shares $M^j_{Y_{op_c}}$ held by peers $\{S_j\}_{j\in\mathcal{T}_i}$ such that $M_{Y_{op_c}} = \sum_{j\in\mathcal{T}_i} M^j_{Y_{op_c}}$ (Y_{op_c} and $M_{Y_{op_c}}$ are specified in detail in Appendix F)

1: S_i computes the masked result in the HE domain:
2: $\quad \text{Enc}_{pk^{(c)}}(Z_{\text{mask}}) = \langle ct,\ Y_{op_c}+M_{Y_{op_c}}\rangle = \text{Enc}_{pk^{(c)}}\left(\langle X, Y_{op_c}\rangle + \langle X, M_{Y_{op_c}}\rangle\right)$
3: S_i also sends ct to the peers $\{S_j\}_{j\in\mathcal{T}_i}$
4: **for** each $j \in \mathcal{T}_i$, in parallel, S_j **do**
5: $\quad$ Compute $\text{Enc}_{pk^{(c)}}(z^j) \triangleq \langle ct,\ M^j_{Y_{op_c}}\rangle$
6: $\quad$ Send $\text{Enc}_{pk^{(c)}}(z^j)$ to S_i
7: **end for**
8: S_i aggregates $\text{Enc}_{pk^{(c)}}(z) = \sum_{j\in\mathcal{T}_i} \text{Enc}_{pk^{(c)}}(z^j) = \text{Enc}_{pk^{(c)}}(\langle X, M_{Y_{op_c}}\rangle)$
9: S_i removes the mask term: $\text{Enc}_{pk^{(c)}}(\langle X, Y_{op_c}\rangle) = \text{Enc}_{pk^{(c)}}(Z) = \text{Enc}_{pk^{(c)}}(Z_{\text{mask}}) - \text{Enc}_{pk^{(c)}}(z)$

Output: $\text{Enc}_{pk^{(c)}}(\langle X, Y_{op_c}\rangle)$

We implement this by homomorphically adding the plaintext-encoded mask:

$$ct' \leftarrow \mathit{Add}\big(ct, \text{Enc}_{pk}(r,\Delta)\big), \qquad \text{Dec}_{sk}(ct')/\Delta = \tilde{x}.$$

Proposition H.1 (Mask-induced distortion)**.** *Let $x \in \mathbb{R}$ be represented in CKKS at scale Δ. Sample $r \xleftarrow{\$} \{-R, -R+1, \ldots, R\}$ and define the revealed masked value $\tilde{x} \triangleq x + r/\Delta$. Then the additive masking introduces a perturbation bounded by*

$$|\tilde{x} - x| \ = \ \left|\frac{r}{\Delta}\right| \ \le \ \frac{R}{\Delta}.$$

Proof. By definition, $\tilde{x} - x = r/\Delta$. Taking absolute values gives $|\tilde{x} - x| = |r|/\Delta$. Since $r \in \{-R, \ldots, R\}$, we have $|r| \le R$, and therefore $|\tilde{x} - x| \le R/\Delta$. □

H.2 Multiplicative Masking with a Bounded Error Budget

Let $x \in \mathbb{R}$ be a decoded plaintext value satisfying $|x| \le B_x$, where $B_x > 0$, and let $\Delta > 0$ be the CKKS scale. We use a randomized multiplicative mask close to one:

$$r \triangleq 1 + \frac{k}{\Delta}, \qquad k \xleftarrow{\$} \{-K, -K+1, \ldots, K\},$$

where K is a nonnegative integer. The parameter K controls the amount of discrete randomization, while the bound below controls its numerical effect. This mask is intended to introduce bounded random perturbation; the following result is an error bound, not a privacy guarantee (see Remark L.6 in Appendix L for the privacy-side implication of this same quantity).

We apply the plaintext-encoded mask at scale Δ and rescale the product:

$$ct' \leftarrow \mathit{PMult}(ct, \text{Enc}_{pk}(r,\Delta)), \qquad ct'' \leftarrow \mathit{Rescale}(ct'),$$

where the rescaling prime p is chosen so that $\text{scale}(ct'') \approx \Delta$ (e.g., $p \approx \Delta$).

Proposition H.2 (Mask-induced perturbation bound)**.** *Let $\varepsilon_{\text{mask}} > 0$ be the error budget allocated to multiplicative masking. Choose integers $K_{\min}, K$ such that*

$$1 \le K_{\min} \le K \le K_{\max} \triangleq \min\left\{ \left\lfloor \frac{\varepsilon_{\text{mask}}\Delta}{B_x} \right\rfloor, \lceil\Delta\rceil - 1 \right\},$$

and sample k uniformly from $\{-K, \ldots, K\}$. Then $r > 0$ and

$$|xr - x| \le \varepsilon_{\text{mask}}.$$

For vector inputs, sample the mask independently for each coordinate. The same bound then holds coordinate-wise whenever $\|X\|_\infty \le B_x$.

Proof. Since $|k| \le K \le \lceil \Delta \rceil - 1 < \Delta$, we have $r = 1 + k/\Delta \ge 1 - K/\Delta > 0$. Moreover,

$$|xr - x| = \frac{|xk|}{\Delta} \le \frac{B_x K}{\Delta} \le \varepsilon_{\text{mask}},$$

where the last inequality follows from $K \le \lfloor \varepsilon_{\text{mask}} \Delta / B_x \rfloor$. Applying the scalar argument to each coordinate proves the vector claim. □

The choice of $K_{\min}$ specifies a minimum randomization level (for example, $K_{\min} = 1$ requires a non-constant mask). The interval is feasible only when $K_{\min} \le K_{\max}$. Since k is uniform, its entropy is $\log_2(2K + 1)$ bits; a target entropy can therefore be used to choose $K_{\min}$. The bound above covers only mask-induced perturbation. CKKS encoding, multiplication, and rescaling errors must be accounted for separately in the total numerical-error budget.

I Masked Non-Linear Evaluation

Our non-linear evaluations rely on *three* reusable primitives built on threshold decryption with masking; detailed protocol descriptions are provided in Appendix J. For simplicity, we **omit** the superscript (c) in $\text{Enc}_{pk^{(c)}}$ in the following, as our evaluation approach is independent of any specific key and applicable to general-purpose settings. The primitives include:

(1) *Exponential* evaluation protocol: $\mathbf{P}_{\text{exp}}$ (Alg. 6), which applies $\phi = \exp(\cdot)$ to the masked plaintext $X + M$, encrypts the result, and removes the mask's contribution through multiplication with an encrypted correction factor derived from M (here $g = 1/\exp(\cdot)$), yielding $\text{Enc}_{pk}(\exp(X))$.

(2) *Reciprocal* evaluation protocol: $\mathbf{P}_{\text{inv}}$ (Alg. 7), which applies $\phi = 1/(\cdot)$ to the masked value XM, obtains the encrypted result, and removes the multiplicative mask's contribution to recover $\text{Enc}_{pk}(1/X)$, where $g = \text{I}$ is the identity function.

(3) *Inverse square root* evaluation protocol: $\mathbf{P}_{\text{invsqrt}}$ (Alg. 8) where $\phi = 1/\sqrt{(\cdot)}$, which follows similar steps to Alg. 7 but employs different multiplicative masking: instead of masking with M, we mask X with M^2. Specifically, the peer servers first generate $[[M^2]] = [[M]] \otimes [[M]]$, then compute $[[Z]] = [[X]] \otimes [[M^2]]$ to obtain $Z = X \cdot M^2$ as in Eq. (4). The generation of $\{M_j^2\}_{j \in \mathcal{T}_i}$ can be entirely performed during the *preprocessing phase*, as M^2 is query-independent, making the online computation cost equivalent to the multiplicative masking in the $1/(\cdot)$ protocol. At each peer server, M_j is sampled from the distribution of $\sqrt{M}/(N_S - 1)$.

Using these three primitives, Softmax, LayerNorm, and SiLU are realized through composition of multiple primitive calls, as detailed in Appendix K

J Non-linear evaluation Primitives

Algorithm 6 $\mathbf{P}_{\text{exp}}$: e^X evaluation protocol

Input: Ciphertext $\text{Enc}_{pk}(X)$

1: Decrypting $\text{Enc}_{pk}(X) \to X + M$ by running collective decryption (additive) (Eq. (2))
2: After applying $\exp(\cdot)$ to $X + M$, encrypting $\exp(X + M)$ to obtain $\text{Enc}_{pk}(\exp(X + M))$
3: Computing $\text{Enc}_{pk}(\frac{1}{\exp(M)})$ through $\prod_{j \in \mathcal{T}_i} \text{Enc}_{pk}(\frac{1}{\exp(M_j)})$, where $\text{Enc}_{pk}(\frac{1}{\exp(M_j)})$ is computed by S_j and sent back to S_i, for $j \in \mathcal{T}_i$, respectively
4: Multiplying $\text{Enc}_{pk}(\exp(X + M))$ by $\text{Enc}_{pk}(\frac{1}{\exp(M)})$ to obtain $\text{Enc}_{pk}(\exp(X))$

Output: $\text{Enc}_{pk}(\exp(X))$

K Non-linear Activation Evaluations

This section gives explicit algorithmic descriptions of the three non-linear evaluations used by SHIM. Each algorithm composes homomorphic operations with the reusable masked exponential, reciprocal,

Algorithm 7 $\mathbf{P}_{\text{inv}}$: $1/X$ evaluation protocol

Input: Ciphertext $\text{Enc}_{pk}(X)$

1: Decrypting $\text{Enc}_{pk}(X) \to XM$ by running collective decryption (multiplicative) (Eq. (4))
2: After applying $\frac{1}{(\cdot)}$ to XM, encrypting $\frac{1}{XM}$ to obtain $\text{Enc}_{pk}(\frac{1}{XM})$
3: Computing $\text{Enc}_{pk}(M)$ through $\sum_{j\in\mathcal{T}_i} \text{Enc}_{pk}(M_j)$, where $\text{Enc}_{pk}(M_j)$ is computed by S_j and sent back to S_i, for $j \in \mathcal{T}_i$, respectively
4: Multiplying $\text{Enc}_{pk}(\frac{1}{XM})$ by $\text{Enc}_{pk}(M)$ to obtain $\text{Enc}_{pk}(\frac{1}{X})$

Output: $\text{Enc}_{pk}(\frac{1}{X})$

Algorithm 8 $\mathbf{P}_{\text{invsqrt}}$: $1/\sqrt{X}$ evaluation protocol

Input: Ciphertext $\text{Enc}_{pk}(X)$

1: Decrypting $\text{Enc}_{pk}(X) \to XM^2$ by running collective decryption (multiplicative) (Eq. (4))
2: After applying $\frac{1}{\sqrt{(\cdot)}}$ to XM^2, encrypting $\frac{1}{\sqrt{X}M}$ to obtain $\text{Enc}_{pk}(\frac{1}{\sqrt{X}M})$
3: Computing $\text{Enc}_{pk}(M)$ through $\sum_{j\in T} \text{Enc}_{pk}(M_j)$, where $\text{Enc}_{pk}(M_j)$ is computed by S_j and sent back to S_i, for $j \in \mathcal{T}_i$, respectively
4: Multiplying $\text{Enc}_{pk}(\frac{1}{\sqrt{X}M})$ by $\text{Enc}_{pk}(M)$ to obtain $\text{Enc}_{pk}(\frac{1}{\sqrt{X}})$

Output: $\text{Enc}_{pk}(\frac{1}{\sqrt{X}})$

and inverse-square-root primitives. The computation server never receives the unmasked activation values.

For notation, we use $pk = pk^{(c)}$ for the current client session and write $ct = \text{Enc}_{pk}(X)$; this is the same encryption notation denoted by $\text{Enc}_{pk^{(c)}}(\cdot)$ elsewhere. The symbol $\odot$ denotes ciphertext–ciphertext multiplication (*CMult*). For a session assigned to S_i, the peer set is $\mathcal{T}_i = \{S_j : j \neq i\}$ and $|\mathcal{T}_i| = N_S - 1$. The decoded additive mask is denoted vectorwise by $\mathbf{m} = \mathbf{r}/\Delta$, consistent with Appendix H.1, where Δ is the CKKS scale. The positive multiplicative mask M is the mask denoted by r in Appendix H.2; the inverse-square-root primitive uses its square M^2. In LayerNorm, d is the vector width, $\boldsymbol{\gamma}$ and $\boldsymbol{\beta}$ are plaintext affine parameters, and $\epsilon > 0$ is the numerical stabilizer.

Softmax Evaluation (Alg. 9). The additive-mask primitive exposes only the masked vector $\widetilde{X} = X + \mathbf{m}$ to the computation server (invoking Eq. (2)). After max shifting, the computation server evaluates $\exp(\widetilde{X} - a\mathbf{1})$ and the peers provide the encrypted correction $\text{Enc}_{pk}(\exp(-\mathbf{m}))$. Therefore,

$$\text{Enc}_{pk}(\exp(X + \mathbf{m} - a\mathbf{1})) \odot \text{Enc}_{pk}(\exp(-\mathbf{m})) = \text{Enc}_{pk}(\exp(X - a\mathbf{1})) \ . \tag{18}$$

Let $E = \text{Enc}_{pk}(\exp(X - a\mathbf{1}))$ and $d_a = \sum \exp(X - a\mathbf{1})$. The computation server computes $D = \sum E = \text{Enc}_{pk}(d_a)$ and invokes the masked reciprocal primitive $\mathbf{P}_{\text{inv}}$ (Alg. 7). If the reciprocal primitive uses a positive multiplicative mask M, it reveals only $\widetilde{d}_a = d_a M$ and returns

$$\text{Enc}_{pk}(1/(d_a M)) \odot \text{Enc}_{pk}(M) = \text{Enc}_{pk}(1/d_a) \ . \tag{19}$$

Finally, multiplying E by this reciprocal yields the desired vector output:

$$E \odot \text{Enc}_{pk}(1/d_a) = \text{Enc}_{pk}\left(\frac{\exp(X)}{\sum \exp(X)}\right) \ . \tag{20}$$

The max shift is computed from the masked vector and is common to every element, so it cancels in the Softmax normalization. In particular, for $\widetilde{X} = X + \mathbf{m}$, $a = \max(\widetilde{X})$, and $\widehat{X} = \widetilde{X} - a\mathbf{1}$, we have $\widehat{X} \leq 0$ element-wise and hence $0 < \exp(\widehat{X}) \leq 1$ element-wise. With the additive mask $\mathbf{m} = \mathbf{r}/\Delta$, $\|\mathbf{r}\|_\infty \leq R$, and Δ denoting the CKKS scale, the shifted range satisfies

$$\max(\widetilde{X}) - \min(\widetilde{X}) \leq W_x + \frac{2R}{\Delta} \ , \qquad W_x = \max(X) - \min(X) \ . \tag{21}$$

Algorithm 9 $\mathbf{P}_{\text{MSoftmax}}$: Masked Softmax Evaluation (composition of $\mathbf{P}_{\text{exp}}$ and $\mathbf{P}_{\text{inv}}$)

Input: Encrypted logits $ct = \text{Enc}_{pk}(X)$

1: The peers collectively reveal to the computation server only $\widetilde{X} = X + \mathbf{m}$ (Eq. (2))
2: Compute $a = \max(\widetilde{X})$ locally and set $\widehat{X} = \widetilde{X} - a\mathbf{1}$
3: Invoke $\mathbf{P}_{\text{exp}}$ (Alg. 6) on the max-shifted input to obtain $ct_E = \text{Enc}_{pk}(\exp(X - a\mathbf{1}))$; the masked evaluation and correction are handled inside $\mathbf{P}_{\text{exp}}$.
4: Compute $ct_D = \sum ct_E = \text{Enc}_{pk}(\sum \exp(X - a\mathbf{1}))$ by the corresponding HE reduction.
5: Invoke the masked reciprocal primitive $\mathbf{P}_{\text{inv}}$ (Alg. 7) on ct_D, which results in $ct_R = \text{Enc}_{pk}(1/\sum \exp(X - a\mathbf{1}))$.
6: $ct_O = ct_E \odot ct_R = \text{Enc}_{pk}(\text{Softmax}(X))$.

Output: $\text{Enc}_{pk}(\text{Softmax}(X)) = ct_O$

Algorithm 10 $\mathbf{P}_{\text{MLayerNorm}}$: Masked LayerNorm Evaluation (composition with $\mathbf{P}_{\text{invsqrt}}$)

Input: Encrypted vector $ct = \text{Enc}_{pk}(X)$, affine parameters $\boldsymbol{\gamma}, \boldsymbol{\beta}$, numerical constant $\epsilon > 0$

1: Compute $ct_\mu = \text{Enc}_{pk}(\mu)$ homomorphically, where $\mu = \text{Mean}(X)$
2: Compute $ct_V = \text{Enc}_{pk}(V)$ homomorphically, where $V = \text{Mean}((X - \mu)^2) + \epsilon$
3: Invoke the masked inverse-square-root primitive $\mathbf{P}_{\text{invsqrt}}$ (Alg. 8) on ct_V to obtain $ct_R = \text{Enc}_{pk}(1/\sqrt{V})$
4: $ct_Y = \text{Enc}_{pk}(X - \mu) \odot ct_R \odot \boldsymbol{\gamma} + \boldsymbol{\beta}$

Output: $\text{Enc}_{pk}(\text{LayerNorm}(X)) = ct_Y$

LayerNorm Evaluation (Algorithm 10). The evaluated implementation computes the mean and variance term homomorphically rather than recovering them from an additively masked activation. For vector X, the computation server obtains

$$\mu = \text{Mean}(X) \ , \qquad V = \text{Mean}((X - \mu)^2) + \epsilon \ , \tag{22}$$

as encrypted values. The inverse-square-root protocol $\mathbf{P}_{\text{invsqrt}}$ (Alg. 8) is invoked only for the non-polynomial inverse-square-root operation. The peers generate a positive multiplicative mask M^2 and collective masked decryption reveals only $\widetilde{V} = VM^2$. The computation server evaluates and re-encrypts

$$\frac{1}{\sqrt{\widetilde{V}}} = \frac{1}{\sqrt{VM^2}} = \frac{1}{M\sqrt{V}} \ . \tag{23}$$

After applying the encrypted correction $\text{Enc}_{pk}(M)$, the computation server obtains

$$\text{Enc}_{pk}\left(\frac{1}{M\sqrt{V}}\right) \odot \text{Enc}_{pk}(M) = \text{Enc}_{pk}\left(\frac{1}{\sqrt{V}}\right) \ . \tag{24}$$

Centering, normalization, and the affine transformation remain homomorphic:

$$\text{Enc}_{pk}(X - \mu) \odot \text{Enc}_{pk}\left(\frac{1}{\sqrt{V}}\right) \odot \boldsymbol{\gamma} + \boldsymbol{\beta} \ . \tag{25}$$

SiLU Evaluation (Algorithm 11). The computation server uses the identity

$$\text{SiLU}(X) = X \odot \frac{\exp(X)}{\mathbf{1} + \exp(X)} \ . \tag{26}$$

The masked exponential primitive $\mathbf{P}_{\text{exp}}$ (Alg. 6) produces $ct_E = \text{Enc}_{pk}(\exp(X))$. The computation server then forms $ct_A = \mathbf{1} + ct_E$ and invokes the masked reciprocal primitive $\mathbf{P}_{\text{inv}}$ (Alg. 7) to obtain $ct_R = \text{Enc}_{pk}(1/(\mathbf{1} + \exp(X)))$. Consequently,

$$\text{Enc}_{pk}(X) \odot ct_E \odot ct_R = \text{Enc}_{pk}\left(X \odot \frac{\exp(X)}{\mathbf{1} + \exp(X)}\right). \tag{27}$$

This construction composes the exponential and reciprocal primitives with HE arithmetic; it does not replace SiLU with a polynomial approximation. Because CKKS arithmetic is approximate, we refer to these operations as non-polynomial rather than exact nonlinear evaluation.

Algorithm 11 $\mathbf{P}_{\text{MSiLU}}$: Masked SiLU Evaluation (composition of $\mathbf{P}_{\text{exp}}$ and $\mathbf{P}_{\text{inv}}$)

Input: Encrypted pre-activation $ct = \text{Enc}_{pk}(X)$
1: Invoke the masked exponential primitive $\mathbf{P}_{\text{exp}}$ (Alg. 6) to obtain $ct_E = \text{Enc}_{pk}(\exp(X))$.
2: Compute $ct_A = ct_E + \text{Enc}_{pk}(\mathbf{1}) = \text{Enc}_{pk}(\mathbf{1} + \exp(X))$
3: Invoke the masked reciprocal primitive $\mathbf{P}_{\text{inv}}$ (Alg. 7) to obtain $ct_R = \text{Enc}_{pk}(1/(\mathbf{1} + \exp(X)))$
4: Compute $ct_\sigma = ct_E \odot ct_R = \text{Enc}_{pk}(\exp(X)/(\mathbf{1} + \exp(X)))$
5: $ct_Y = ct \odot ct_\sigma = \text{Enc}_{pk}(X \odot \exp(X)/(\mathbf{1} + \exp(X)))$
Output: $\text{Enc}_{pk}(\text{SiLU}(X)) = ct_Y$

K.1 Numerical Stability of Exponential and Softmax

The computation server applies max-shifting directly to the masked vector before invoking the exponential primitive. Let $\widetilde{X} = X + \mathbf{m}$ denote the value revealed by the additive-masking protocol. The server computes

$$a = \max(\widetilde{X}) \ , \qquad \widehat{X} = \widetilde{X} - a\mathbf{1} \ . \tag{28}$$

This operation is local and does not require a secure-maximum protocol or an additional communication round. Since $\widehat{X} \le 0$ element-wise, it guarantees

$$0 < \exp(\widehat{X}) \le 1 \ , \tag{29}$$

which prevents overflow in the exponential evaluation. The additive correction remains valid after shifting because the common shift cancels during normalization:

$$\frac{\exp(X + \mathbf{m} - a\mathbf{1}) \odot \exp(-\mathbf{m})}{\sum(\exp(X + \mathbf{m} - a\mathbf{1}) \odot \exp(-\mathbf{m}))} = \frac{\exp(X - a\mathbf{1})}{\sum \exp(X - a\mathbf{1})} = \frac{\exp(X)}{\sum \exp(X)} \ . \tag{30}$$

For the additive mask $\mathbf{m} = \mathbf{r}/\Delta$, where $\|\mathbf{r}\|_\infty \le R$ and Δ is the CKKS scale, we have $\|\mathbf{m}\|_\infty \le R/\Delta$. If $W_x = \max(X) - \min(X)$ is the unmasked logit spread, then

$$\max(\widetilde{X}) - \min(\widetilde{X}) \le W_x + \frac{2R}{\Delta} \ , \qquad \widehat{X} \in \left[-\left(W_x + \frac{2R}{\Delta} \right), 0 \right] \ . \tag{31}$$

Thus, the max-shift controls the input range of the exponential primitive independently of the absolute magnitude of the logits. This range-control argument is separate from CKKS encoding precision and does not change the underlying approximate-arithmetic semantics of CKKS.

L Security Analysis

We restate the assumptions from Sec. 3.3 that the following analysis relies on: CKKS is IND-CPA secure under Ring-LWE; additive secret sharing provides $(n-1)$-privacy with $n = N_S - 1$; the MPC multiplication protocol is secure against semi-honest adversaries; smudging noise statistically hides partial-decryption shares; and at most $N_S - 2$ of the N_S servers are corrupted with a fixed corruption set and semi-honest behavior throughout. This is the sharing privacy threshold used in our analysis, not a general honest-majority bound.

Scope and leakage profile. Consistent with the threat model in Sec. 3.3, our guarantees protect the confidentiality of the client's plaintext query, intermediate activations, KV cache entries, and generated tokens against any coalition $\mathcal{A}$ of at most $N_S - 2$ corrupted servers. This is a guarantee about *client content*, not about the model: it does not by itself hide the model architecture from the servers that run it.

We first isolate two logically separate quantities: (a) the *realized decoding length* $t \le \tau_{\max}$ at which a given session stops, and (b) the *shape* of the protocol transcript: how many times each of $\mathbf{P}_{\text{refresh}}$, $\mathbf{P}_{\text{nonlinear}}$, $\mathbf{P}_{\text{KVM}}$, $\mathbf{P}_{\text{unmaskmul}}$ fires and at which (l, t, op_c), *given* that length. These behave very differently with respect to content-dependence.

t itself is content-dependent and we do not try to hide it: Alg 1's decoding loop runs `while` $t < \tau_{\max}$ `and not` $\langle$`EOS`$\rangle$, so a short factual query and an open-ended generation task will, in

general, realize different t, and any observer of the transcript learns t (equivalently, how many tokens the response took, though not what they are). This is a standard, low-severity leakage shared by essentially all private-LLM-serving systems in this space, including the FHE and hybrid baselines we compare against, and we do not claim otherwise; we also do not hide the static cap $\tau_{\max}$ or the model architecture and layer count L.

What Lemmas L.1–L.4 rule out is any *further* leakage *conditioned on* t. For a fixed model and a *given* realized length t, every triggering condition in Algorithm 1 (`if` $\ell_{ct} = 0$; `if` op_c is non-linear; `if` a new KV pair is generated; `if` op_c is an attention product) branches on the ciphertext level and the static circuit $\mathcal{C}$ both of which are fixed functions of (l, t) under leveled HE, and never on the decrypted value of X_{input} or any intermediate activation. So, among sessions that happen to realize the *same* t, the transcript shape is identical regardless of the specific tokens involved: two clients who both end up generating, say, exactly 37 tokens produce indistinguishable invocation patterns, even though their prompts and outputs differ completely. This holds for whichever party observes the transcript.

Whether that transcript shape constitutes leakage *beyond* t and the architecture, however, depends on who is observing it and what they already know. In ***SHIM*** (Sec. 3.1), N_S servers can be operated by *distinct entities*, and only the servers authorized as computation servers are guaranteed to hold a model copy; per Sec. 3.1, "helper-only servers can participate without receiving the model weights." A peer server S_j acting purely as a helper for client session c is told, invocation by invocation, *which* primitive it must execute, e.g., S_j returns only $\text{Enc}(M_j)$ for $\mathbf{P}_{\text{refresh}}$, but must additionally compute $\text{Enc}(1/\exp(M_j))$ inside $\mathbf{P}_{\text{exp}}$ (used by Softmax/SiLU) or generate M_j^2 instead of M_j inside $\mathbf{P}_{\text{invsqrt}}$ (used by LayerNorm). This is more than an opaque timing signal: it identifies *which* non-linear function is being evaluated and at what layer/step cadence. If such an S_j does not already know the architecture, this reveals genuine structural information about it (roughly the layer count, the placement of Softmax/LayerNorm/SiLU, and the KV-cache growth rate) that S_j would not otherwise have.

To be precise about *who* this affects, recall from Sec. 3.1–C that $1 \le N_{\text{comp}} \le N_S$ denotes the number of servers authorized to hold a model copy in a given deployment, and that role assignment is per-session: whichever server is not the compute server for session c is, for that session, a peer that must observe and act on this signal, means there is no way to participate in the threshold protocol *without* being told which primitive to run. So every peer, model-holding or not, watches the same messages. The distinction that matters is whether that observation teaches the peer anything it did not already know:

- If $N_{\text{comp}} = N_S$ (every server holds a synchronized model copy, e.g., under the fully parallel deployment of Sec. 3.1), then a server acting as a peer for another server's session already possesses the complete architecture from its own local copy, independent of and prior to observing anything in that session's transcript. Watching which primitive fires for someone else's session is then *redundant*: it adds zero bits about the architecture beyond what the peer already had. (It does still learn session-specific facts about c that are leaked regardless of N_{comp}, such as c's realized length t, which is the leakage discussed above, but not new architectural information.)
- If $N_{\text{comp}} < N_S$ (e.g., $N_{\text{comp}} = 1$, only one server has the GPU and model), the $N_S - N_{\text{comp}}$ helper-only servers never hold an independent copy of the architecture. For them, watching the same transcript is not redundant: it is the *only* channel through which they ever learn anything about that architecture, and it does teach them genuine structural information (roughly the layer count, the placement of Softmax/LayerNorm/SiLU, and the KV-cache growth rate).

In practice, then, this residual channel is benign whenever $N_{\text{comp}} = N_S$, or whenever the base architecture is already public independent of the protocol regardless of N_{comp}, as in our evaluation on the open-weight LLaMA-3.1-8B-Instruct, whose architecture is public knowledge to any party, participating or not. It is *not* closed by the cryptographic core of ***SHIM*** whenever $N_{\text{comp}} < N_S$ and the architecture is proprietary, since Theorem L.5 and Lemmas L.1–L.4 provide a client-content confidentiality guarantee under the stated assumptions; they do not protect the model structure implied by which primitive a helper-only peer is asked to run. We list this explicitly as a limitation of the present analysis rather than folding it into the "benign public metadata" framing, since that framing

implicitly assumes every party already holds the model (as in a design where both servers always host it) and does not automatically transfer to *SHIM*'s distinct-entity, partial-model-replication deployment model.

Side-channel leakage and model-level inference attacks (e.g., membership inference) remain outside the scope of this cryptographic analysis.

L.1 Security of the Masked-Threshold-Decryption Primitives

Lemma L.1 (Security of Masked Threshold Decryption)**.** *Let $ct = Enc_{pk^{(c)}}(X)$ denote any ciphertext held by S_i that is submitted to threshold decryption with additive masking: each peer S_j, $j \in \mathcal{T}_i$, computes $d_j = PDec_{sk_j^{(c)}}(ct) = c_1 \cdot sk_j^{(c)} + e_{sm,j} \bmod Q$ (Eq. 1), samples an independent additive mask share M_j from the distribution of Appendix H.1 scaled by $1/(N_S - 1)$, and sends $(d_j + M_j,\ Enc_{pk^{(c)}}(M_j))$ to S_i, who reconstructs $X + M = c_0 + \sum_{j\in\mathcal{T}_i}(d_j + M_j) \bmod Q$ (Eq. 2). This is exactly the step executed by:*

- ***P**$_{refresh}$ (Alg. 2), with X the ciphertext being refreshed;*
- *the additive-masking step inside **P**$_{nonlinear}$ (Alg. 3), with X the pre-activation input to **P**$_{exp}$; and*
- ***P**$_{KVM}$ (Alg. 4), applied independently with $X = Q_t^{(l)}$ and $X = A_t^{(l)}$ at every layer l and decoding step t.*

Under the assumptions above, for any coalition $\mathcal{A}$ of at most $N_S - 2$ corrupted servers, the view of $\mathcal{A}$ during this step is simulatable from public parameters alone; in particular, it reveals nothing about X.

Proof. We case-split on whether $S_i \in \mathcal{A}$, restricted to this single invocation.

Case 1: $S_i \notin \mathcal{A}$. Then $\mathcal{A}$ controls at most $N_S - 2$ of the $N_S - 1$ peers in $\mathcal{T}_i$, so at least one peer is honest. By $(n-1)$-privacy of additive secret sharing Cramer et al. (2015) with $n = N_S - 1$, the shares $\{sk_j^{(c)}\}_{j\in\mathcal{A}}$ are statistically independent of $sk^{(c)}$, so $\mathcal{A}$ cannot recover the key; the discrete-Gaussian smudging noise $e_{sm,j}$ additionally hides the secret-key-dependent term in each d_j that $\mathcal{A}$ observes Li et al. (2023); Froelicher et al. (2023). Because at least one mask share M_j (from the honest peer) is unknown to $\mathcal{A}$ and is sampled independently of X, the aggregate mask $M = \sum_{j\in\mathcal{T}_i} M_j$ is uniform and unknown to $\mathcal{A}$; hence the reconstructed value $X + M$, which is computed only by the honest S_i, is never observed by $\mathcal{A}$, and the shares $\{d_j + M_j\}_{j\in\mathcal{A}\cap\mathcal{T}_i}$ that $\mathcal{A}$ *does* observe are themselves uniformly random from $\mathcal{A}$'s view by the same $(n-1)$-privacy argument.

Case 2: $S_i \in \mathcal{A}$. Then $\mathcal{A}$ controls S_i plus at most $N_S - 3$ peers, leaving at least 2 peers honest. S_i observes the reconstructed $X + M$, but since $M = \sum_{j\in\mathcal{T}_i} M_j$ includes at least two independently sampled mask shares unknown to $\mathcal{A}$, M is statistically uniform from $\mathcal{A}$'s view and acts as a one-time pad, so $X + M$ reveals nothing about X. S_i also does not hold any share of $sk^{(c)}$, so it cannot decrypt ct on its own; the encrypted mask terms $\mathrm{Enc}_{pk^{(c)}}(M_j)$ reveal nothing further by IND-CPA security of CKKS.

In both cases $\mathcal{A}$'s view is simulatable given only the fact that a masking step occurred (public per the leakage profile above), completing the proof. Substituting $X \leftarrow Q_t^{(l)}$ or $X \leftarrow A_t^{(l)}$ recovers the $\mathbf{P}_{\mathrm{KVM}}$ instance; the argument is otherwise unchanged, since $\mathbf{P}_{\mathrm{KVM}}$ never removes the mask (the cache is stored *as* $X + M$), which only strengthens the claim. □

Remark L.2 (Immunity to correlated smudging-noise leakage). Hwang et al. Hwang et al. (2025) identify a subtle simulation-based issue in general CKKS-based MPC: if a single party both samples the smudging noise for a partial decryption *and* later observes the correlated masked-decryption output, the two correlated views can leak more than the smudging noise alone is designed to hide. This pattern does not arise in SHIM's masking step. In Lemma L.1, the smudging noise $e_{sm,j}$ for a given invocation is sampled and consumed entirely by peer S_j, who never receives the reconstructed value $X+M$: only S_i aggregates $\{d_j+M_j\}_{j\in\mathcal{T}_i}$ into $X+M$, and S_i never learns any individual $e_{sm,j}$ (it is folded into d_j before transmission). Thus no single server ever simultaneously holds: (a) the smudging

randomness it generated; and (b) the correlated masked output derived from it. Moreover, every invocation of $\mathbf{P}_{\text{refresh}}$, $\mathbf{P}_{\text{nonlinear}}$, and $\mathbf{P}_{\text{KVM}}$ draws fresh, independent $e_{sm,j}$ and M_j (Appendix H.1), so this separation holds invocation-by-invocation and cannot be bridged across invocations by a single corrupted party. We therefore treat the risk identified in Hwang et al. (2025) as addressed by construction for the semi-honest, honest-majority setting analyzed here, rather than out of scope.

L.2 Security of Multiplicative Masking with MPC

Lemma L.3 (Security of Multiplicative Masking)**.** *Consider the multiplicative-masking step used inside $\boldsymbol{P}_{nonlinear}$ by the reciprocal primitive $\boldsymbol{P}_{inv}$ (Alg. 7) and the inverse-square-root primitive $\boldsymbol{P}_{invsqrt}$ (Alg. 8): peers hold additive shares $[[X]] = \{X_j\}_{j\in\mathcal{T}_i}$ and $[[M]] = \{M_j\}_{j\in\mathcal{T}_i}$ (or $[[M^2]]$ for $\boldsymbol{P}_{invsqrt}$), jointly compute $[[Z]] = [[X]] \otimes [[M]]$ via a secure multiplication protocol to obtain $X \cdot M$ (Eq. 3–4), and each peer sends its result share together with an encrypted correction term $\text{Enc}_{pk^{(c)}}(g(M_j))$ to S_i. Under the same assumptions as Lemma L.1, this step reveals nothing about X, M, or Z to any coalition $\mathcal{A}$ of at most $N_S - 2$ corrupted servers.*

Proof. The proof rests on the same three ingredients used throughout this section: privacy of the peers' additive shares, semi-honest security of the MPC multiplication sub-protocol, and IND-CPA security of the encrypted correction terms:

If $S_i \notin \mathcal{A}$: by the same $(n-1)$-privacy argument as Lemma L.1 (here $n = N_S - 1$), $\mathcal{A}$'s shares $\{X_j, M_j\}_{j\in\mathcal{A}}$ are independent of X and M; the secure-multiplication sub-protocol is secure against semi-honest adversaries Damgård et al. (2013), so $\mathcal{A}$'s view during the $\otimes$ computation is simulatable from its own input/output shares alone, without learning X, M, or $Z = X \cdot M$; and S_i, being honest, never exposes the reconstructed $X \cdot M$ to $\mathcal{A}$.

If $S_i \in \mathcal{A}$: S_i observes $X \cdot M = \sum_{j\in\mathcal{T}_i} Z_j$, but since $M = \sum_{j\in\mathcal{T}_i} M_j$ includes at least two mask shares from honest peers, M is uniform and unknown to $\mathcal{A}$, so $X \cdot M$ acts as a one-time-pad-style multiplicative mask that reveals nothing about X (this is exactly the multiplicative analogue of the one-time-pad argument in Lemma L.1); the corrupted peers' views during $\otimes$ remain simulatable by the same MPC security argument, and the encrypted corrections $\text{Enc}_{pk^{(c)}}(g(M_j))$ reveal nothing further by IND-CPA security. □

L.3 Security of the Ciphertext–Masked-Plaintext Product ($\mathbf{P}_{\text{UNMASKMUL}}$)

Lemma L.4 (Security of $\mathbf{P}_{\text{unmaskmul}}$)**.** *Let $ct = \text{Enc}_{pk^{(c)}}(X)$ and let $Y_{op_c} + M_{Y_{op_c}}$ be a masked-plaintext cache operand with shares $\{M^j_{Y_{op_c}}\}_{j\in\mathcal{T}_i}$ held by the peers, as consumed by $\boldsymbol{P}_{unmaskmul}$ (Alg. 5) which is instantiated for attention-logit computation with $X = Q^{(l)}_t$, $Y_{op_c} + M_{Y_{op_c}} = K^{(l)}_{1:t} + M^{(l)}_{K,1:t}$, and for attention-output computation with $X = A^{(l)}_t$, $Y_{op_c} + M_{Y_{op_c}} = V^{(l)}_{1:t} + M^{(l)}_{V,1:t}$. For any coalition $\mathcal{A}$ of at most $N_S - 2$ corrupted servers, including $S_i \in \mathcal{A}$, the protocol reveals nothing about X, Y_{op_c}, or any mask share $M^j_{Y_{op_c}}$ beyond what is implied by ciphertexts exchanged under $pk^{(c)}$.*

Proof. Unlike Lemmas L.1 and L.3, $\mathbf{P}_{\text{unmaskmul}}$ never invokes threshold decryption: every message is a ciphertext under $pk^{(c)}$ where S_i sends ct to the peers, and each peer S_j returns $\text{Enc}_{pk^{(c)}}(z^j) \triangleq \langle ct, M^j_{Y_{op_c}}\rangle$, computed homomorphically as a ciphertext–plaintext inner product without ever decrypting or transmitting $M^j_{Y_{op_c}}$ itself, even in encrypted form as a standalone value. Consequently, security reduces directly to IND-CPA security of CKKS: for any $\mathcal{A}$ (including $S_i \in \mathcal{A}$), every value $\mathcal{A}$ observes, including ct, $\{\text{Enc}_{pk^{(c)}}(z^j)\}_{j\in\mathcal{T}_i\cap\mathcal{A}}$, and the aggregated $\text{Enc}_{pk^{(c)}}(Z_{\text{mask}})$ if $S_i \in \mathcal{A}$, is a ciphertext under $pk^{(c)}$, and $\mathcal{A}$ never holds a share of $sk^{(c)}$ (Sec. 3.3).

Hence, by IND-CPA security, these ciphertexts are computationally indistinguishable from encryptions of independent random plaintexts, so $\mathcal{A}$'s view is simulatable without knowledge of X, Y_{op_c}, or the mask shares $M^j_{Y_{op_c}}$. This also means a corrupted S_i cannot use repeated queries to extract $M^j_{Y_{op_c}}$ and thereby compromise *future* accesses to the masked KV cache: because S_i follows the protocol specification (semi-honest model) and never submits adversarially chosen ciphertexts, this

guarantee holds throughout the session. Substituting $X \leftarrow Q_t^{(l)}$ or $X \leftarrow A_t^{(l)}$ recovers, respectively, the attention-logit instantiation ($Y_{op_c}+M_{Y_{op_c}} = K_{1:t}^{(l)}+M_{K,1:t}^{(l)}$) and the attention-output instantiation ($Y_{op_c}+M_{Y_{op_c}} = V_{1:t}^{(l)}+M_{V,1:t}^{(l)}$); the argument above is otherwise unchanged in both cases. □

L.4 End-to-End Privacy of the SHIM Protocol

Theorem L.5 (Privacy of the Hybrid HE-MPC Protocol)**.** *Consider Algorithm 1 for private transformer inference, where client session c is assigned to compute server S_i and $sk^{(c)}$ is additively shared among the peers $\{S_j\}_{j\in\mathcal{T}_i}$, $\mathcal{T}_i = \{1,\dots,N_S\}\setminus\{i\}$. Under the assumptions listed at the start of this section, for any coalition $\mathcal{A}$ of at most N_S-2 corrupted servers, the view of $\mathcal{A}$ over a full run of Algorithm 1—spanning all L layers and up to $\tau_{\max}$ decoding steps, and including every invocation of $\mathbf{P}_{refresh}$, $\mathbf{P}_{nonlinear}$, $\mathbf{P}_{KVM}$, and $\mathbf{P}_{unmaskmul}$—reveals no information about the client's input query, intermediate plaintexts, KV cache plaintexts, or generated output tokens, beyond the public leakage profile stated above.*

Proof. By construction, Algorithm 1 interleaves four kinds of steps:

(i) Purely homomorphic linear operations on ciphertexts under $pk^{(c)}$;

(ii) Invocations of $\mathbf{P}_{\text{refresh}}$ or the additive-masking step of $\mathbf{P}_{\text{nonlinear}}$, or invocations of $\mathbf{P}_{\text{KVM}}$, all covered by Lemma L.1;

(iii) The multiplicative-masking step inside $\mathbf{P}_{\text{nonlinear}}$'s $\mathbf{P}_{\text{inv}}$/$\mathbf{P}_{\text{invsqrt}}$ calls, covered by Lemma L.3;

(iv) Invocations of $\mathbf{P}_{\text{unmaskmul}}$ for attention-logit and attention-output computation, covered by Lemma L.4.

Step (i) reveals nothing to $\mathcal{A}$ by IND-CPA security alone. Each occurrence of steps (ii)–(iv) is, by the respective lemma, simulatable from public parameters alone, *independent of* X, using fresh randomness (mask shares and smudging noise) sampled anew at every invocation (Appendix H.1, H.2). The only values that persist across invocations and cross the interface between them are ciphertexts under $pk^{(c)}$ (for step (i)) or masked-plaintext KV cache entries whose masks remain secret-shared among the peers and are never reconstructed by S_i (for $\mathbf{P}_{\text{KVM}}$ outputs consumed later by $\mathbf{P}_{\text{unmaskmul}}$); no protocol step reveals a mask share itself. Hence the simulators for steps (i)–(iv) can be composed: at each invocation, the simulator for that step is invoked on the (simulatable) ciphertext-valued state produced by all prior steps, and produces output computationally indistinguishable from the real execution.

Thus, by the composition theorem for secure protocols Canetti (2020), a sequential composition of simulatable sub-protocols whose only shared interface is ciphertexts (or un-reconstructed secret-shared masks) is itself simulatable. Applying this across all L layers and all $t = 1,\dots,\tau_{\max}$ decoding steps of Algorithm 1 shows that $\mathcal{A}$'s view of the entire run is simulatable from public parameters alone, establishing the claim for both $S_i \notin \mathcal{A}$ and $S_i \in \mathcal{A}$. □

Remark L.6 (Concrete masking-parameter margin). The one-time-pad arguments in Lemmas L.1 and L.3 require only that the aggregate mask M be uniform over a range unknown to $\mathcal{A}$; we do not otherwise fix the range parameters R (additive masking in Appendix H.1) or K (multiplicative masking in Appendix H.2). These parameters play a dual role. Propositions H.1 and H.2 bound R/Δ and the error budget $\varepsilon_{\text{mask}}$ as the *numerical distortion* introduced by masking, but the same quantity is, from $\mathcal{A}$'s perspective, exactly the residual precision to which X is determined by observing $X+M$ or $X\cdot M$.

Consequently, The fact that M is unknown to $\mathcal{A}$ implies that $X+M$ (or $X\cdot M$) reveals nothing about X only when R (resp. K) is chosen sufficiently large relative to the assumed bound on X (resp. B_x), such that the residual information is cryptographically negligible rather than merely small for accuracy purposes; minimizing R or K to reduce numerical distortion, as Appendix H.1–H.2 otherwise motivates, pulls in the opposite direction.

Remark L.7 (Removing the trusted dealer). The analysis above, and Appendix C, assume a trusted key generator that creates and distributes $\{pk^{(c)}, \mathbf{evk}^{(c)}, sk^{(c)}\}$ and then deletes all key material. This is a deployment convenience, not a requirement of the protocol itself: any of the theorem and lemmas above go through unchanged if $\{pk^{(c)}, \{sk_j^{(c)}\}_{j\in\mathcal{T}_i}\}$ are instead produced by a standard Distributed Key Generation protocol Mouchet et al. (2021) run by S_i and the peers, so that no single party, even including the key generator, ever observes $sk^{(c)}$ in the clear. We leave a full instantiation with DKG to future work, but note it removes the single trusted party in Appendix C entirely.