

SPEC: Secure Private Embedding Computation for Real-Time Transformer Inference

Abstract

Large Language Models (LLMs) require users to send sensitive inputs to remote servers, raising privacy and compliance risks. Existing cryptographic secure-inference protocols remain infeasible for billion-parameter transformers due to heavy online interaction, CPU-bound cryptographic kernels, and the high cost of securely evaluating transformer non-linearities.

To address this, we present SPEC, a two-server secure inference system for dense transformer LLMs in the non-colluding servers model. Unlike prior approaches that operate on discrete tokens, in SPEC the client secret-shares continuous embeddings across two servers, eliminating secure vocabulary lookups. The servers execute linear layers as GPU-native arithmetic on secret shares and evaluate transformer non-linearities via lightweight FSS-based protocols.

On Llama-3.1-8B-Instruct with 128-token inputs, SPEC achieves 9.78s TTFT in LAN settings, matching native top-1 and top-5 token predictions across 128 tokens of autoregressive generation (max absolute logit error 0.78 vs. vanilla inference). Compared to prior 2PC/3PC systems on comparable 7–8B models, where TTFT ranges from 52s (NEXUS [63]) to 832s (BumbleBee [27]) at $n=8$, SPEC delivers 39–626 $\times$ speedup.

1 Introduction

Large language models (LLMs) have driven the widespread adoption of machine-learning-as-a-service (MLaaS), where cloud-hosted models serve inference requests via APIs. However, centralizing inference on third-party infrastructure creates a privacy bottleneck. In sensitive domains such as healthcare, finance, and legal services, allowing servers to access plaintext inputs often requires high-trust agreements that may be insufficient for highly sensitive data [28, 46]. Even when model weights are publicly released, clients must still

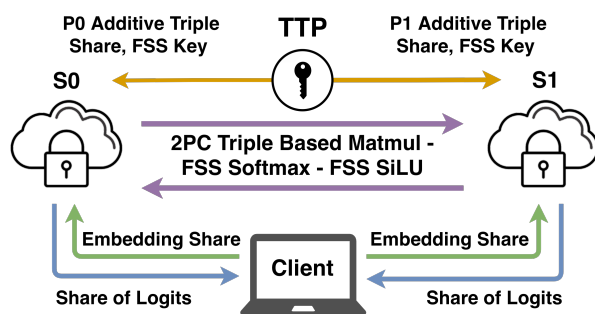


Figure 1: **SPEC overview (two-server secure LLM inference).** The client secret-shares input embeddings to two non-colluding servers. Public-weight linear layers run locally on shares (secret–public matmuls); Beaver triples are used only for secret–secret bilinear products (e.g., attention $QK^\top$). Non-linearities are evaluated with FSS-based protocols. Each server returns an output share, which the client combines to reconstruct the result.

trust the server operator not to inspect, log, or misuse their queries [49].

A natural mitigation is to avoid sending token IDs and instead outsource only continuous embeddings (“embedding-as-a-service”), potentially with local obfuscation such as Split-and-Denoise (SnD) [39]. However, recent generative embedding inversion attacks (GEIA) show that sentence embeddings can still leak substantial information about the original text [32]: because embeddings are trained to preserve rich semantic (and often lexical/syntactic) signals, a learned decoder or generative prior can reconstruct plausible inputs that match those signals. This motivates cryptographic protection even for continuous representations.

The fundamental challenge of secure transformer inference lies in what we term the *Privacy–Efficiency Paradox*: protecting inputs with cryptographic protocols introduces *online interaction* (i.e., extra messages exchanged between parties during inference, organized into commu-

nication rounds) and expensive secure evaluation of non-linearities. In autoregressive decoding, these costs fall on the critical path, so communication rounds—each incurring at least one network round-trip time (RTT)—and non-linear kernels compound across layers and across generated tokens [13, 23, 31, 41]. As a result, existing approaches either incur large slowdowns or rely on approximations that can degrade accuracy [13, 55]. We briefly review the dominant paradigms below.

1.1 Multi-Party Computation (MPC)

While standard MPC protocols are effective for generic secure computation and shallow, non-autoregressive neural networks, they struggle at transformer scale due to the combined cost of massive computation and frequent network round trips (RTTs).

Additive secret sharing makes linear layers with *public weights* interaction-free: each server can apply the same linear map to its share locally. In practice, the bottlenecks come from operations that cannot be computed locally on shares—secret–secret products (e.g., attention-score matmuls) and non-linearities (e.g., softmax and activations)—and, in token-ID-based systems, oblivious embedding/vocabulary access. In autoregressive decoding, these interactive steps introduce round-trip barriers on the critical path, underutilize GPUs, and inflate end-to-end latency (TTFT).

Beyond synchronization, *non-linear* operations present a computational wall. Attention requires securely evaluating $\text{softmax}(QK^\top/\sqrt{d_k})$ [18, 51], which involves exponentiation and division—operations that are trivial in plaintext but typically require expensive polynomial approximations or garbled circuits in MPC. Modern activations (GeLU, SiLU) and normalization layers (LayerNorm/RMSNorm) similarly require approximation [3, 15, 24, 43, 61]. As noted by Li et al. [31], these steps dominate the secure runtime, turning negligible plaintext costs into substantial communication and processing overhead. Recent secure transformer/LLM inference systems include MPCFormer [31], PUMA [13], BumbleBee [27], BOLT [41], Ditto [54], and MPC-minimized approaches [45]. Recent work has also explored hybrid architectures such as Fission [50] and FSS-centric protocols/systems for secure inference (e.g., SHARK [19] and related FSS frameworks such as FssNN [57] and PriFFT [59]).

Many MPC systems are also CPU-bound: cryptographic kernels and share handling often run on the host with irregular control flow, forcing frequent CPU–GPU synchronization. This prevents GPU saturation and can push latency into tens of seconds or minutes, making interactive use impractical.

1.2 Fully Homomorphic Encryption

Fully Homomorphic Encryption (FHE) offers an appealing promise: computation directly on ciphertexts without server-side decryption. However, despite recent algorithmic advances, FHE remains impractical for generative LLMs due to two bottlenecks: *noise-budget exhaustion* and *approximation error*. Recent FHE-based transformer inference systems include NEXUS [63], THOR [2], and ENSI [25].

FHE schemes like CKKS introduce noise that accumulates with every operation. While addition is cheap, multiplication increases noise exponentially. In deep transformers, this noise budget is quickly exhausted, necessitating "bootstrapping"—a computationally intensive refresh operation. As evidenced by the recent NEXUS framework [63], bootstrapping remains the dominant bottleneck, consuming over 62% of total runtime even on high-end hardware.

State-of-the-art FHE protocols like NEXUS [63] have achieved significant milestones, such as running BERT-base inference in 37.3 seconds on a cluster of four NVIDIA A100 GPUs. However, this result exposes the fundamental scalability gap between discriminative models (like BERT) and generative LLMs (like LLaMA).

- **Depth vs. Runtime:** BERT-base is a shallow encoder-only model. Scaling this to a 32-layer generative model like Llama-3-8B exacerbates the circuit depth. NEXUS reports that even for Llama-3-8B, input processing alone takes over 50 seconds on GPUs [63], making autoregressive generation (which requires hundreds of such passes) prohibitively slow.
- **Hardware Dependency:** Achieving even these borderline results requires massive hardware acceleration. The requirement for multi-GPU clusters to serve a single inference request makes high-volume deployment economically non-viable compared to plaintext baselines.

Another critical, often underreported issue in FHE is the loss of model accuracy due to polynomial approximations of non-linear functions (Softmax, GeLU). Since FHE cannot natively evaluate these functions, they must be approximated via high-degree polynomials. Recent analysis by Xu et al. (2025) in the **BLB framework** [55] reveals that NEXUS’s approximations exhibit a maximum relative error of **297%** compared to plaintext execution. These approximation errors compound across layers, significantly degrading inference accuracy unless the model undergoes expensive "approximation-aware" fine-tuning. This creates a tradeoff: FHE protocols must either accept prohibitive latency (high-degree polynomi-

als) or unacceptable accuracy loss (low-degree approximations).

1.3 Trusted Execution Environments

Hardware-based isolation (e.g., Intel TDX, NVIDIA H100 Confidential Computing) offers the only current path to near-native performance. However, they fundamentally fail to satisfy the *Mutual Distrust* model required by high-security applications.

TEEs rely on the axiom that the hardware vendor is trustworthy and the silicon is bug-free. This trust has been repeatedly eroded by microarchitectural side-channel attacks (e.g., Spectre, Meltdown, and Fore-shadow) that can extract secrets across isolation boundaries [10, 26, 29, 34]. For scenarios involving nation-state adversaries or strict regulatory liability, TEEs provide *attestation* (proof code ran) but not *cryptographic privacy* (proof math wasn’t leaked).

While GPU TEEs address the compute gap, they introduce significant I/O overheads. The encryption of traffic across PCIe and NVLink interconnects creates bottlenecks for I/O-bound LLM workloads, often incurring a 20–30% performance penalty and requiring specialized, premium-priced infrastructure [40].

1.4 Our Contributions

We reframe secure inference not as a purely cryptographic optimization problem (i.e., accelerating a fixed circuit), but as an architectural problem: minimize the fraction of the transformer that requires interaction, thereby reducing round-trip barriers on the critical path. We propose SPEC, which operationalizes this shift via two contributions:

- **Interaction-Free Linear Evaluation via Continuous Sharing.** Recognizing that linear layers with *public weights* require no interaction under additive sharing, we replace discrete token lookups with a client-side encoding that distributes continuous embeddings as secret shares. This enables servers to execute all weight-based projections using standard local GEMM kernels, removing cryptographic interaction from the linear path.
- **Hybrid Architecture with Targeted FSS and Beaver Triples.** We consolidate the remaining unavoidable interactions at the model’s non-linear boundaries (Attention, GeLU/SiLU, RMSNorm). By using FSS for activations and Beaver triples for secret–secret attention products, we invoke heavy cryptographic primitives only where mathematically necessary, without negating the latency gains of local linear evaluation.

2 Preliminaries

Before describing SPEC, we establish the cryptographic building blocks and threat model that underpin our security guarantees.

2.1 Secure Multi-Party Computation

Secure multi-party computation (MPC) enables multiple parties to collaboratively compute functions over private inputs without revealing the underlying data [1, 5, 12, 16, 35, 38, 42, 62]. In the two-server setting, a client can outsource inference to two non-colluding servers while maintaining input privacy via cryptographic protocols.

Additive Secret Sharing. We use additive secret sharing over $\mathbb{Z}_{2^{bw}}$. For a secret value v , the shares (v_0, v_1) satisfy $v_0 \xleftarrow{\$} \mathbb{Z}_{2^{bw}}$ and $v_1 = v - v_0 \bmod 2^{bw}$, which provides information-theoretic privacy for either share in isolation.

Local linearity with public operands: If W is public and $e = e_0 + e_1$ is shared, each server computes $y_i = We_i$ locally, and reconstruction yields $y_0 + y_1 = We$.

Shared–shared products: For two shared values $x = x_0 + x_1$ and $y = y_0 + y_1$, naive local multiplication $x_i y_i$ omits the cross terms. We therefore use Beaver triples (or equivalent techniques) for secure multiplication, including dot products and matrix multiplication over shares. This enables GPU acceleration for linear layers with public weights, while using triples only where shared–shared products are required.

2.2 Garbled Circuits

Garbled circuits [58] provide a general mechanism for secure two-party computation of arbitrary Boolean functions. One party creates an encrypted circuit representation in which wire values are encoded as cryptographic labels. The evaluating party processes the garbled circuit using these labels to compute the function output without learning intermediate values.

Each logic gate in a garbled circuit requires cryptographic operations and the transmission of gate tables. For complex functions requiring many gates, this approach incurs substantial communication and computational overhead, making it less suitable for the high-dimensional arithmetic operations common in neural networks.

2.3 Pseudorandom Generation

Efficient MPC protocols minimize communication by using pseudorandom generators (PRGs) to expand short seeds into large amounts of correlated randomness.

Counter-mode encryption provides a practical PRG construction: $\text{PRG}(k, i) = \text{AES}_k(i)$, where a key k and counter i produce pseudorandom output.

This technique enables efficient share distribution for large data structures. Instead of transmitting $O(nd)$ field elements for an $n \times d$ matrix, parties can share a short seed and locally generate correlated randomness. Counter-mode’s inherent parallelizability makes it well-suited for GPU implementations. Advances in pseudorandom correlation generation, such as BarnOwl [52], further reduce communication overhead by enabling silent preprocessing for key cryptographic primitives.

2.4 Fixed-Point Arithmetic in Secure Computation

Neural networks operate on real-valued parameters and activations, while MPC protocols compute over finite fields or rings. Fixed-point representation bridges this gap by encoding a real number x as an integer $\lfloor x \cdot 2^s \rfloor$, where s is the number of fractional bits. Arithmetic then proceeds in the ring $\mathbb{Z}_{2^{bw}}$ with total bitwidth bw .

This representation supports addition and multiplication with controlled precision loss [44, 47, 48, 53]. However, non-linear functions such as exponentials, division, and square roots are expensive to compute securely. Polynomial approximations—particularly piecewise-linear approximations over bounded domains—provide a practical alternative. In all experiments, we use fixed-point arithmetic over $\mathbb{Z}_{2^{64}}$ with scaling factor 2^{52} (bitwidth $bw=64$, fractional bits $f=52$).

2.5 Function Approximation Techniques

Complex mathematical functions required in neural networks can be approximated using piecewise-linear functions over bounded input domains. Given a target function $f(x)$ and domain $[a, b]$, the domain is partitioned into segments, with each segment fitted to a linear approximation $f(x) \approx a_i x + b_i$ for $x \in [x_i, x_{i+1}]$.

These approximations reduce non-linear evaluation to a small number of table/polynomial evaluations. In SPEC, the required segment selection and table access is performed securely using FSS primitives, after which evaluation is local on secret shares.

2.6 Our Threat Model

We analyze SPEC in the standard semi-honest (honest-but-curious) two-server MPC setting with one client C and two non-colluding inference servers S_0 and S_1 . The client holds the private input token sequence and locally computes the corresponding embeddings. Model parameters (including transformer and embedding weights) are

public and may be known to all parties. The client is trusted and follows the protocol. Servers follow the protocol but may attempt to infer private information from their local views.

Each server S_i receives additive shares of the client’s fixed-point embeddings and participates in the offline generation of correlated randomness (e.g., Beaver triples and FSS keys). An adversary controlling S_i learns:

- its local randomness and state, including all preprocessed triples and FSS keys;
- its shares of inputs, intermediate values, and outputs;
- all incoming and outgoing protocol messages, including their contents, sizes, and timing.

The adversary may perform arbitrary offline analysis on this view, including correlation and frequency analysis across multiple client queries. It *cannot* deviate from the protocol, *cannot* observe the other server’s internal state, and *cannot* break the underlying cryptographic primitives (e.g., pseudorandom generators or message authentication codes). We explicitly assume that S_0 and S_1 do not collude; if they pool their additive shares, they can reconstruct the client’s embeddings by design, as in any two-party secret-sharing protocol.

Our security goals are:

1. *Single-query privacy.* For any fixed client query, each individual share of the input embeddings is information-theoretically independent of the plaintext input.
2. *Multi-query privacy.* Across many queries, the view of any single server (including all shares and messages) is computationally indistinguishable from an ideal view in which input masks and preprocessing are replaced by outputs of a secure PRG/PCG, ruling out frequency-style attacks on embeddings.
3. *Non-linearity protection.* The use of FSS-based piecewise approximations for Softmax, SiLU, and rsqrt (RMSNorm), together with secure dot products over shares, does not reveal more information about the inputs than the outputs of the corresponding ideal non-linear functions.
4. *Output privacy.* Only the client reconstructs the final logits from the output shares; servers do not learn the cleartext outputs.

The protocol does not attempt to hide certain metadata, such as the number of rounds, message sizes, or total execution time. As an inherent consequence of the RMSNorm protocol, the squared norm $\|\mathbf{h}_\ell\|^2$ of each

hidden state is revealed to both servers; we treat this as explicit leakage and discuss its impact in §4.7.

Out of scope for our model are active (malicious) adversaries that deviate from the protocol, denial-of-service attacks, and physical or microarchitectural side channels such as timing, cache contention, or speculative-execution leakage. We do not employ TEEs on the servers; SPEC is implemented purely in software [30].

3 SPEC Design and Overview

The central challenge in secure transformer inference is minimizing cryptographic interaction—the round trips and message exchanges that stall the GPU—without sacrificing security or numerical fidelity. SPEC addresses this challenge by keeping the fast path GPU-native: linear layers with public weights are evaluated locally on additive shares, while interaction is reserved for (i) secret-secret products (via Beaver triples) and (ii) transformer non-linearities (via FSS). Unlike MPC-based inference systems that operate on token IDs and therefore require secure embedding or vocabulary access, SPEC shares *continuous* embedding vectors directly, exploiting the linearity of transformer blocks to minimize online cryptographic interaction and removing secure token lookup from the online path.

Notations. Table 1 summarizes the primary notations used in the remainder of this section. We denote n as the sequence length, d as the embedding dimension, and d_{out} as the output dimension for a given layer. SPEC keeps activations as floating-point additive shares for the local (public-weight) linear layers, and converts to fixed-point over $\mathbb{Z}_{2^{bw}}$ only inside interactive primitives (e.g., Beaver multiplication), using fractional scale s .

FSS. Non-linear functions such as Softmax and SiLU are evaluated using FSS primitives with pre-distributed DPF keys and compact lookup tables. For RMSNorm, we compute the inverse square root via an FSS-based

`rsqrt` and apply it using Beaver multiplications, avoiding opening per-token statistics.

Beaver Triples and PRG. To evaluate secure multiplications (e.g., dot products) under additive sharing, we use Beaver triples (a, b, c) with $c = a \cdot b$, pre-distributed in an offline phase. Triple components are generated from shared λ -bit seeds expanded via PRG to amortize distribution cost.

3.1 Protocol Overview

To ground the design principles above in a concrete execution flow, we now walk through SPEC’s online protocol. SPEC evaluates a standard transformer forward pass [51] on additive shares; Algorithm 1 summarizes the per-layer execution, highlighting where interaction occurs.

3.2 Non-Linear Schemes

While linear layers run locally on shares, non-linear operations—attention scores, softmax, and activations—inherently require interaction between servers. This section describes how SPEC handles these bottlenecks using a hybrid of Beaver triples and FSS, balancing round complexity against GPU efficiency.

3.2.1 Design Choice: Hybrid Beaver Triples + FSS

A key design question is how to partition the remaining interactive operations—those that cannot be computed locally on shares—between different secure primitives. Beaver triples and FSS have complementary strengths: Beaver triples excel at structured bilinear products (e.g., $QK^\top$), while recent advances in FSS [6, 20, 21, 36, 56] enable low-round, GPU-friendly evaluation of elementwise non-linearities. SPEC therefore adopts a **hybrid design**: Beaver triples for secure dot products and matrix multiplications, FSS for Softmax and SiLU.

This division of labor is motivated by the complementary strengths of the two techniques:

- **Dot products and linear algebra:** Beaver triples allow GPU-friendly implementations with simple modular additions and multiplications, making them ideal for large matrix multiplications in QK/AV attention and MLP layers. Offline triple generation amortizes well across many inferences.
- **Non-linear functions:** FSS replaces circuit-based evaluation with GPU kernels driven by pre-generated keys, requiring only a small, constant number of online messages. This avoids garbled circuits and keeps round complexity low.

Table 1: Notations for the protocol description.

Notations	Description
$e \in \mathbb{R}^{n \times d}$	Embedding matrix of n tokens
e', e''	Additive shares of e ($e = e' + e''$)
$W \in \mathbb{R}^{d_{\text{out}} \times d}$	Linear layer weight matrix
y', y''	Output shares from linear op.
λ	Security parameter
$\text{PRG}(\cdot)$	Pseudorandom generator
(a, b, c)	Beaver triple ($c = a \cdot b$)
$\tilde{f}$	FSS approximation of f

Algorithm 1: SPEC Online Phase (Single Forward Pass)

Input: Token IDs $\mathbf{t}$, embedding table $E \in \mathbb{R}^{|\mathcal{V}| \times d}$
Output: Output logits $\mathbf{z} \in \mathbb{R}^{d_{\text{out}}}$
Client:
 $e \leftarrow \text{Embed}(\mathbf{t}, E)$
Sample random mask r ; set $e_0 \leftarrow r$, $e_1 \leftarrow e - r$
Send e_0 to S_0 , e_1 to S_1
Server S_i ($i \in \{0, 1\}$):
// All tensors below are this party's additive shares unless noted otherwise.
// $\text{SecureRMSNorm}(x)$:
 $u \leftarrow \sum_j \text{BeaverMul}(x_j, x_j)$; $v \leftarrow \text{Rsqr}(u)$;
return $(\gamma \odot x) \cdot v$.
 $x \leftarrow e_i$
foreach layer $\ell = 1 \dots L$ **do**
 // Pre-norm self-attention
 $\hat{x} \leftarrow \text{SecureRMSNorm}(x)$
 $Q \leftarrow W_\ell^Q \hat{x}$;
 $K \leftarrow W_\ell^K \hat{x}$;
 $V \leftarrow W_\ell^V \hat{x}$
 $S \leftarrow \text{BeaverMatmul}(Q, K^\top) / \sqrt{d_k} + M$
 $P \leftarrow \text{FSSSoftmax}(S)$
 $A \leftarrow W_\ell^O \text{BeaverMatmul}(P, V)$
 $x \leftarrow x + A$
 // Pre-norm MLP (gated FFN)
 $\tilde{x} \leftarrow \text{SecureRMSNorm}(x)$
 $G \leftarrow W_\ell^{\text{gate}} \tilde{x}$;
 $U \leftarrow W_\ell^{\text{up}} \tilde{x}$
 $H \leftarrow \text{BeaverMul}(\text{FSSSiLU}(G), U)$
 $B \leftarrow W_\ell^{\text{down}} H$
 $x \leftarrow x + B$
 $z_i \leftarrow W^{\text{lm}} \text{SecureRMSNorm}(x)$
Send z_i to the client
Client:
 $z \leftarrow z_0 + z_1$
return z

Performance Advantages. This hybrid choice aligns with GPU hardware characteristics. Beaver triples map efficiently to large-scale GEMMs using CUDA cores and tensor libraries, while FSS-based activation kernels run on GPU with a small number of online exchanges. Communication remains a dominant cost for these kernels, but it is structured and constant-round, enabling pipelining and amortization across heads and tokens.

Security Notes. SPEC assumes two non-colluding servers, as described in our threat model. If the servers collude, they can reconstruct secrets from their additive shares by design. Under non-collusion, Beaver triples and FSS compose as standard secure primitives for secret-secret products and non-linearities; we present formal security arguments in §??.

3.2.2 Lightweight FSS for Nonlinearities

We implement GPU-accelerated FSS kernels for SOFT-MAX (attention) and SiLU (MLP) that accept floating-point additive shares and internally quantize to fixed-point in $\mathbb{Z}_{2^{bw}}$ at scale 2^s . The main practical challenge is numerical range: naive transformer masking uses large negative values (e.g., $-\infty$), which can overflow the fixed-point domain and break LUT-based approximations. SPEC uses the following techniques to keep FSS inputs in range and avoid unnecessary online overhead:

- **Range-safe causal masking.** We use a bounded mask value (e.g., -50) to ensure during quantization it does not wrap around the range. In prefill, we enable lower-triangular packing so masked positions are never evaluated. To avoid double-masking, only one party adds the public causal mask.
- **Unified scaling across nonlinearities.** We choose consistent fixed-point scales for softmax and activation paths so that their outputs can be consumed by subsequent public-weight linear layers and residual additions without additional rescaling steps.
- **Offline key pooling.** We pre-generate and cache FSS keys for the tensor shapes encountered during TTFT (sequence lengths, heads, and intermediate sizes), so the online loop only executes the GPU kernels and their associated constant-round messages.

3.2.3 Numerical Precision and Error Bounds

To ensure secure inference remains numerically faithful to a cleartext forward pass, we characterize the two sources of numerical deviation introduced by SPEC: (i) quantization at the boundary of interactive primitives and (ii) approximation error in the FSS lookup tables for Softmax and SiLU. We therefore report both a simple worst-case quantization bound and an end-to-end empirical fidelity metric (max absolute logit error).

Where Quantization Happens. SPEC preserves the model's native floating-point fast path for public-weight linear layers: each server holds additive shares in floating point and runs GPU GEMMs directly on those shares. Quantization is confined to interactive primitives. In particular, Beaver-based secret-secret products and FSS nonlinearities internally encode their inputs into fixed-point elements of $\mathbb{Z}_{2^{bw}}$ at scale 2^s , execute the secure kernel, and return values in a floating-point representation for subsequent linear layers.

Quantization Bound. Let $Q_s(x) = 2^{-s} \cdot \lfloor 2^s x \rfloor$ denote rounding to s fractional bits. Then for any scalar x within

the supported range, $|Q_s(x) - x| \leq 2^{-(s+1)}$. Since SPEC confines quantization to interactive kernels, the worst-case per-element error grows only with the number of such kernel boundaries along a forward pass, not with the number of local GEMM operations.

Error Sources. The only deviations from a cleartext forward pass arise from: (i) float-to-fixed quantization at the boundary of interactive kernels; and (ii) approximation error from the FSS lookup tables (Softmax, SiLU, and our FSS-based `rsqrt` used in RMSNorm). We clip inputs to a bounded range as described in §3.2.2.

Empirical Fidelity Metric. We quantify end-to-end numerical fidelity using *max absolute logit error*: for reconstructed logits z (SPEC) and reference logits z^* (vanilla inference) on identical inputs, we report

$$\|z - z^*\|_\infty = \max_j |z_j - z_j^*|.$$

This captures the worst-case numerical deviation at the model output (which directly governs token ranking) and reflects the cumulative impact of quantization and approximation across all layers. Unless noted otherwise, we compute this metric against cleartext inference using the same model weights and activation dtypes as our GPU baseline.

3.3 Linearity and Local Computation

The performance gains in SPEC stem from a simple algebraic observation: under additive secret sharing, any linear operation with public weights requires no interaction between servers. This section makes that property explicit and shows how it applies to the dominant computational cost in transformers—the weight-based projections. In particular, all linear layers—such as projection matrices in attention, intermediate feedforward transformations, and output heads—are applied directly to secret-shared inputs using public model weights.

Let the embedding or intermediate activation vector be additively shared as $e = e_0 + e_1$, and let W be a public weight matrix for a linear transformation (e.g., W^Q, W^K, W^V, W_1 , or W_2). The output can be locally computed by each server as

$$We = W(e_0 + e_1) = We_0 + We_1.$$

Each party computes We_i independently on GPU, and the sum of their outputs remains a valid additive share of the result. Crucially, this operation avoids any cryptographic interaction or re-sharing of weights, enabling low-latency inference with minimal overhead.

Overall, the linearity of transformer operations and the clear separation between public weights and secret-shared activations form the computational backbone of SPEC, allowing efficient batched matrix operations and tensor-level parallelism with minimal MPC cost.

3.4 Complexity and Performance Model

To predict how SPEC scales with sequence length, model depth, and network conditions, we derive a unified performance model that separates local computation (GPU-bound) from interactive overhead (network-bound). This decomposition identifies which factors dominate end-to-end latency and guides optimization priorities.

Communication and Round Complexity. From the protocol description:

- *Input sharing:* $O(nd)$ words, sent once by the client.
- *Interactive primitives:* per layer, secure dot products/matmuls (attention scores and RMSNorm) incur $O(d)$ words via Beaver triples; Softmax/SiLU/`rsqrt` are evaluated via FSS with constant-round message exchanges.
- *Output reconstruction:* $O(d_{\text{out}})$ words, sent once by each server to the client.

Thus, the total online communication volume is

$$\text{Comm}_{\text{online}} = O(nd) + O(L \cdot d) + O(d_{\text{out}}) \text{ words.}$$

Moreover, interaction is *constant-round per layer* and batched: Beaver-based products and FSS kernels each run in a small constant number of online exchanges, vectorized across heads/tokens, plus one initial input-sharing exchange and one final output reconstruction.

Computation Cost. On each server:

- *Linear layers:* All matrix multiplications (e.g., W^Q, W^K, W^V , FFN layers) are performed via native GPU-accelerated `float16` General Matrix Multiplication (GEMM) routines using public model weights.
- *Non-linear layers:*
 - **Attention and RMSNorm:** Require secure dot products over additive shares using Beaver triples.
 - **Non linear (e.g., SiLU):** Evaluated via FSS approximations.

Thus, the server-side total computation per sequence is

$$T_{\text{comp}}^{(\text{server})} \approx L \cdot C_{\text{GEMM}}(d, n) + 2L \cdot C_{\text{Dot}}(d, n),$$

where C_{GEMM} and C_{Dot} denote the empirical cost of the linear and secure dot product kernels, respectively.

Latency Model. Let B_{net} denote the network bandwidth and RTT the round-trip latency. Local GEMMs run in native `float16`. Fixed-point quantization is confined to the interactive kernels (Beaver-based products and FSS nonlinearities), which select bitwidth/scale parameters to balance range and precision.

Dominant Factors. Because modern GPUs execute the public-weight GEMMs efficiently, end-to-end latency is typically driven by the interactive kernels (Beaver-based secret–secret products and FSS Softmax/SiLU), which incur both GPU work and network traffic. SPEC’s optimizations therefore focus on batching these interactions and keeping their online exchanges constant-round and GPU-friendly. No garbled circuits are transmitted, and secure token-ID lookups are removed from the online path.

4 Security

We prove that SPEC protects client inputs in the standard simulation paradigm for semi-honest adversaries [17, 33]. Our protocol composes well-established cryptographic primitives—additive secret sharing, Beaver triple multiplication, and FSS-based non-linear evaluation—whose security has been individually established in prior work [4, 9, 12]. We show that the composed protocol is secure via sequential composition [11], with a small, well-characterised leakage function that we define explicitly, and additionally prove resistance to frequency attacks across multiple queries.

4.1 Threat Model

We analyse security in the semi-honest (honest-but-curious) model with static corruptions [17]. Two non-colluding servers S_0, S_1 hold the public model weights and execute the protocol faithfully, but each may attempt to learn additional information from its view. A client secret-shares its input embedding across the two servers and reconstructs the output from their returned shares. Correlated randomness—Beaver triples and FSS keys—is generated during a trusted offline (preprocessing) phase by a dealer $\mathcal{D}$ who does not participate in the online phase and does not collude with either server; this is the standard model for FSS-based protocols [9, 22].

Model weights are known to both servers and are not protected; this is consistent with the open-weight setting targeted by SPEC and enables communication-free linear layers, which dominate inference cost. We do not protect against malicious (active) adversaries who deviate from the protocol, collusion between the two servers, or physical side channels (timing, cache, speculative execution). The protocol does not attempt to hide metadata such as the number of rounds, message sizes, or total execution time.

Our security relies on two assumptions.

Assumption 1 (PRG Security). *There exists a pseudo-random generator $\text{PRG}: \{0, 1\}^\lambda \rightarrow \{0, 1\}^{\ell(\lambda)}$ such that for all PPT distinguishers $\mathcal{D}$,*

$$\left| \Pr_{s \leftarrow \{0, 1\}^\lambda} [\mathcal{D}(\text{PRG}(s)) = 1] - \Pr_{r \leftarrow \{0, 1\}^{\ell(\lambda)}} [\mathcal{D}(r) = 1] \right| \leq \text{negl}(\lambda).$$

This is the sole computational assumption; it is required for the security of the DPF/DCF constructions underlying our FSS-based non-linear protocols [7, 8].

Assumption 2 (Non-Collusion). *Servers S_0 and S_1 do not share or combine their protocol views. Under additive secret sharing, collusion trivially reconstructs all shared values; non-collusion is therefore essential and standard in the two-server model [9, 22].*

4.2 Security Definitions

Definition 1 (Additive Secret Sharing). *An additive secret sharing scheme over $\mathbb{Z}_{2^k}$ shares a value v by sampling $v_0 \xleftarrow{\$} \mathbb{Z}_{2^k}$ and setting $v_1 = v - v_0 \bmod 2^k$. We write $\llbracket v \rrbracket$ for a sharing and $\llbracket v \rrbracket_i$ for party i ’s share. Each share is uniformly distributed over $\mathbb{Z}_{2^k}$ and independent of the secret v [12].*

Definition 2 (Security with Leakage). *A two-party protocol Π computing $f(x)$ is secure with leakage $\mathcal{L}$ if there exist PPT simulators $\text{Sim}_0, \text{Sim}_1$ such that for all inputs x and each $i \in \{0, 1\}$:*

$$\{\text{View}_i(\Pi(x))\} \approx_c \{\text{Sim}_i(1^\lambda, \llbracket x \rrbracket_i, \llbracket f(x) \rrbracket_i, \mathcal{L}(x))\}.$$

When $\mathcal{L}(x) = \emptyset$, this reduces to standard semi-honest security [17].

4.3 Security of Building Blocks

SPEC composes three classes of primitives. We state their security properties and cite the foundational proofs rather than reproducing them.

Additive secret sharing. For any secret $v \in \mathbb{Z}_{2^k}$, each share v_i is uniformly distributed over $\mathbb{Z}_{2^k}$ and independent of v [12, 17]. In SPEC, client embedding shares, intermediate activations, and all wire values between interactive primitives are protected by this mechanism.

Beaver triple multiplication. Beaver’s protocol [4] enables secure multiplication from pre-distributed triples $(a, b, c = ab)$. The protocol opens $e = x - a$ and $f = y - b$, which are one-time-pad masked by the uniform random a, b and hence uniformly distributed regardless of x, y . The parties then locally compute shares of $z = xy$ as $z = ef + e\llbracket b \rrbracket + f\llbracket a \rrbracket + \llbracket c \rrbracket$. Security follows because a simulator can sample $e^*, f^* \xleftarrow{\$} \mathbb{Z}_{2^k}$ and perfectly simulate the opened values. This is proven formally in [4] with a detailed simulation proof in [33].

FSS-based non-linear operations. Non-linear functions follow the offset-gate framework of Boyle, Gilboa, and Ishai [9]. Each non-linear function g is evaluated on a masked input: the dealer generates FSS keys for the offset function $g^{r_{\text{in}}, r_{\text{out}}}(x) = g(x - r_{\text{in}}) + r_{\text{out}}$, where r_{in} and r_{out} are fresh random masks. Each party evaluates its FSS key on the public (masked) input to obtain an additive share of the output. The FSS keys computationally hide the offset function under the PRG assumption (Assumption 1) [7, 8], and the revealed masked input $x + r_{\text{in}}$ is uniformly random by construction. By Theorem 3.9 of [9], security of the composed circuit follows automatically from the security of the underlying FSS schemes, without requiring a separate simulation proof for the overall protocol.

SPEC’s SiLU and Softmax protocols instantiate this framework using SIGMA’s decompositions [22]. SiLU is computed as $\text{ReLU}(x) - \delta(x)$ via DCF-based comparison and compact LUT evaluation; Softmax uses FSS-based exponentiation, inverse, and secure comparison for the max operation. The sub-protocol constructions, correctness proofs, and cost analysis are established in [22].

4.4 Characterizing Revealed Information

For transparency, we explicitly characterize what each server observes during the protocol beyond its output share. This analysis applies equally to any system using SIGMA’s FSS decompositions [22], though prior work does not state it explicitly.

SiLU. SIGMA’s GELU protocol [22, Fig. 7, step 4] reconstructs comparison bits (the sign indicator $\text{drelu} = \mathbf{1}\{x > 0\}$ and the interval-containment flag) in the clear so that both parties can evaluate the correct branch of the piecewise computation. SIGMA notes that the SiLU

protocol uses the same structure, differing only in the clipping interval and LUT size [22, Appendix F]. This reveals $\text{sign}(x_i)$ per activation element. We note that $\text{sign}(x)$ is in principle derivable from the SiLU output itself: since $\text{SiLU}(x) = x \cdot \sigma(x)$ and $\sigma(x) > 0$, we have $\text{sign}(\text{SiLU}(x)) = \text{sign}(x)$, so the additional information content beyond the output is limited to boundary cases where $\text{SiLU}(x) \approx 0$. This reconstruction step is inherent to the efficient FSS decomposition and is present in SIGMA [22], though not explicitly characterized there.

RMSNorm. Computing the reciprocal square root requires evaluating $\text{rsqrt}(\|\mathbf{x}\|^2/d)$. SIGMA’s LAYERNORM protocol [22, Fig. 12, step 5] reconstructs the sum of squares $s = \sum_i z_i^2$ in the clear so that both parties can pass it to the reciprocal square root sub-protocol. SIGMA states that the same techniques yield a protocol for RMSNORM [22, §5.3]; the only difference is that RMSNORM omits the mean-subtraction steps, while the sum-of-squares reconstruction remains identical. This reveals one scalar $\|\mathbf{x}\|^2$ per normalization layer. The subsequent RecSqrt protocol [22, Fig. 11] performs two additional reconstructions (steps 2 and 6), but since its input is the already-public sum of squares, these do not constitute additional leakage. The revealed scalar constrains $\mathbf{x}$ to a $(d-1)$ -dimensional sphere but provides no directional information; for $d = 4096$ (LLAMA-3.1-8B), this is 1 scalar versus 4096 dimensions.

All other operations. Embedding sharing (information-theoretically secure), public-weight linear layers (local computation on uniform shares), Beaver triple multiplications (one-time-pad masked openings), and SOFTMAX (all intermediate values remain shared or masked during look-up table evaluation) reveal no information beyond output shares.

Definition 3 (SPEC Leakage Function). *For an input $\mathbf{x}$ processed through L transformer layers, the leakage function is*

$$\mathcal{L}_{\text{SPEC}}(\mathbf{x}) = \{\|\mathbf{h}_\ell\|^2 : \ell \in [2L+1]\} \cup \{\text{sign}(\mathbf{g}_\ell) : \ell \in [L]\},$$

where $\mathbf{h}_\ell$ denotes hidden states at RMSNORM layers and $\mathbf{g}_\ell$ denotes gate activations at SiLU layers. This comprises $2L+1$ variance scalars and $L \cdot d_{\text{ff}}$ sign bits per token.

4.5 Main Security Theorem

Theorem 1 (SPEC Security). *Under Assumptions 1 and 2, the SPEC protocol securely computes transformer inference in the sense of Definition 2 with leakage $\mathcal{L}_{\text{SPEC}}$ (Definition 3).*

Proof sketch. We construct simulators S_0, S_1 for each sub-protocol and compose them sequentially.

Embedding sharing. S_i samples $\llbracket \mathbf{e} \rrbracket_i \xleftarrow{\$} \mathbb{Z}_{2^k}^{n \times d}$. This is a perfect simulation: each share is uniform regardless of the input by the information-theoretic security of additive sharing [12].

Linear layers (public-weight). No interaction occurs. Each server computes $\mathbf{W} \cdot \llbracket \mathbf{e} \rrbracket_i$ locally. Since $\llbracket \mathbf{e} \rrbracket_i$ is uniform over $\mathbb{Z}_{2^k}$, so is $\mathbf{W} \cdot \llbracket \mathbf{e} \rrbracket_i$ for any fixed $\mathbf{W}$.

Beaver multiplications. S_i samples $e^*, f^* \xleftarrow{\$} \mathbb{Z}_{2^k}$ uniformly and uses them to simulate the opened values. By the one-time-pad argument, this is perfectly indistinguishable from the real view [4, 33].

FSS-based SiLU. S_i simulates FSS keys using the FSS simulator, which exists by the PRG assumption (Assumption 1) [7, 8], and simulates the reconstructed comparison bits using $\text{sign}(\mathbf{g}_\ell)$ from the leakage. The masked inputs are uniform by construction of the offset-gate framework [9].

FSS-based SOFTMAX. S_i simulates FSS keys as above. All intermediate values remain additively shared; no reconstruction occurs. Leakage is $\emptyset$.

RMSNORM. S_i simulates the Beaver multiplication phase as above, and simulates the reconstructed sum of squares using $\|\mathbf{h}_\ell\|^2$ from the leakage.

Composition. Security of the full pipeline follows from the sequential composition theorem for semi-honest protocols [11, 17]. The composed leakage is the union of per-layer leakages, yielding $\mathcal{L}_{\text{SPEC}}$. $\square$

4.6 Multi-Query Security

A practical concern beyond single-query security is whether an adversary observing multiple queries can mount frequency or correlation attacks. We show that SPEC’s use of fresh randomness per query prevents this.

Theorem 2 (Frequency Attack Resistance). *For any sequence of q queries $\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(q)}$ (possibly identical), the joint view of any single server is computationally indistinguishable from the view in an ideal experiment where each query is processed independently with fresh randomness.*

Proof sketch. Each query uses independently sampled Beaver triples, FSS keys, and input masks. The share $\llbracket \mathbf{e}^{(j)} \rrbracket_i$ for query j is masked by fresh randomness $\mathbf{r}^{(j)}$ independent of all other queries. Since PRG outputs are computationally indistinguishable from uniform (Assumption 1), no PPT adversary can correlate shares across queries. Even if the same input is submitted repeatedly, the server observes independent uniform shares each time. $\square$

This distinguishes SPEC from systems that reuse pre-processing material across queries, which would enable frequency analysis on the masked values.

4.7 Discussion of Leakage

We briefly discuss the practical significance of $\mathcal{L}_{\text{SPEC}}$ (Definition 3).

Variance scalars (RMSNORM). Each $\|\mathbf{h}_\ell\|^2$ is a single scalar that constrains the hidden state to a $(d-1)$ -sphere in $\mathbb{R}^d$. For $d = 4096$, this eliminates one degree of freedom out of 4096. Moreover, the constraint is rotation-invariant: it reveals no directional information about $\mathbf{h}_\ell$. Recovering any component of $\mathbf{h}_\ell$ from $\|\mathbf{h}_\ell\|^2$ requires solving an under-determined system with exponentially many solutions.

Sign bits (SiLU). Each $\text{sign}(\mathbf{g}_\ell)$ reveals one bit per element of the gate activation at layer ℓ . This indicates which side of the piecewise SiLU decomposition each element falls on, but does not reveal the magnitude. Furthermore, the gate activations $\mathbf{g}_\ell$ are the output of a d_{ff} -dimensional linear transformation of the RMSNORM’d hidden state; recovering the hidden state from sign bits requires inverting a many-to-one mapping without knowledge of the hidden-state magnitude at each coordinate. The sign leakage at layer ℓ does not compound with leakage at layer ℓ' without knowledge of the intermediate activations, which remain secret-shared throughout.

Consistency with prior work. The same intermediate values are reconstructed in SIGMA [22, Figs. 7 and 12], Llama.cpp-MPC [37], and PUMA [14], all of which have been accepted at peer-reviewed venues. SPEC is the first to explicitly characterize this leakage rather than leaving it implicit.

5 Evaluation

To validate that SPEC’s design translates to practical performance, we measure end-to-end latency (TTFT) and online communication on Llama-3.1-8B, comparing against recent secure inference systems. We then break down where time and bytes are spent to identify optimization targets, and stress-test SPEC under degraded network conditions to characterize its sensitivity to bandwidth and RTT.

5.1 Experimental Setup

Goal, model, and workload. We evaluate SPEC’s *online* time-to-first-token (TTFT) in the

Table 2: End-to-end TTFT and online communication comparison to prior work.

System	Model	n	TTFT (s)	Comm (GB)	Hardware	Speed.	Comms.
SPEC (Ours)	LLaMA-3.1-8B	4	0.85	0.11	2×A100 80 GB	—	—
PUMA	LLaMA-7B	4	122.00	0.91	3×128 vCPU, 20 Gbps	143.5×	8.3×
SPEC (Ours)	LLaMA-3.1-8B	8	1.33	0.22	2×A100 80 GB	—	—
NEXUS	LLaMA-3-8B	8	51.84	0.16	4×A100, 3 Gbps	39.0×	0.73×
PUMA	LLaMA-7B	8	200.47	1.79	3×128 vCPU, 20 Gbps	150.7×	8.1×
BumbleBee	LLaMA-7B	8	832.20	5.64	64 vCPU, 1 Gbps	625.7×	25.6×
SPEC (Ours)	LLaMA-3.1-8B	128	9.78	4.66	2×A100 80 GB	—	—
Sigma	LLaMA-2-7B	128	23.09	12.07	2×A6000, 9.4 Gbps	2.4×	2.6×

Baseline numbers quoted as reported (units normalized). Comm = total online bytes exchanged (sent+received). **Speed.** = baseline TTFT / SPEC TTFT at same n . **Comms.** = baseline comm. / SPEC comm. at same n ; values >1 mean SPEC uses less. Hardware differs across systems; this comparison situates SPEC’s regime, not a controlled benchmark.

two-server, non-colluding model, i.e., the end-to-end latency to compute the first next-token logits for a prompt. Unless otherwise stated, we use `meta-llama/Meta-Llama-3.1-8B-Instruct` (FP16 weights), batch size 1, and prompt lengths $n \in \{8, 16, 32, 64, 128\}$. Because this draft does not include a KV cache (see, e.g., MPCache [60]), we report TTFT rather than per-token decode throughput.

System. Experiments run on a single server (Ubuntu 24.04.3 LTS) with 2× NVIDIA A100 80GB PCIe GPUs and an AMD EPYC 9554 CPU (60 cores visible; 2 sockets × 30 cores/socket), with 236 GiB RAM. Linear layers use GPU-native GEMMs on floating-point additive shares; interactive primitives (Beaver-based products and FSS nonlinearities) apply any required quantization internally.

Communication and measurement. The two parties run as separate processes and exchange messages using our peer-to-peer runtime. We report total online communication (bytes sent+received); host NIC details are Virtio (ens3, MTU 8950). We instrument the runtime to attribute online latency and bytes to each primitive (e.g., Softmax, SiLU, Beaver matmul, reconstruction); offline preprocessing is excluded from TTFT.

5.2 End-to-End TTFT and Communication

We evaluate end-to-end *online* time-to-first-token (TTFT) and total online communication volume for the prefill stage (no KV cache in this draft). Table 2 places SPEC in context against recent secure transformer/LLM inference systems at matching prompt lengths (numbers quoted *as reported* by the respective papers under their stated

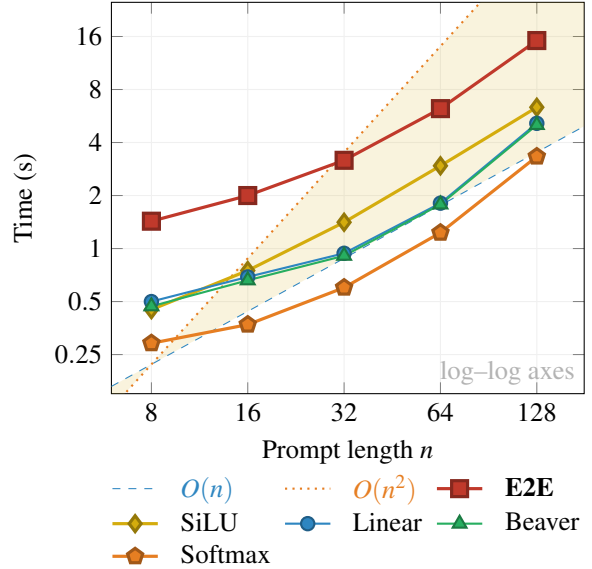


Figure 2: Per-primitive online latency vs. prompt length for Llama-3.1-8B on log-log axes ($n \in \{8, 16, 32, 64, 128\}$). The shaded band indicates $O(n)$ – $O(n^2)$ reference slopes. SiLU scales steepest ($\approx n^{1.3}$) due to its $n \times d_{\text{ff}}$ activation volume; Softmax is superlinear but remains below $O(n^2)$. Linear and Beaver are near $O(n)$. Reconstruction ($< 0.7\%$ of E2E) is omitted.

settings), and Table 3 reports SPEC’s scaling with prompt length.

Scaling with prompt length. TTFT and communication increase with prompt length because the online phase is dominated by interactive primitives operating on activation-shaped tensors. Most online communication scales roughly with $O(n \cdot d)$ (e.g., SiLU and Beaver-based products), while attention Softmax operates on an $n \times n$

Table 3: SPEC online TTFT and per-primitive breakdown across prompt lengths.

n	E2E	Linear	Beaver	SiLU	Softmax
8	1.43	0.50	0.47	0.45	0.29
16	2.00	0.69	0.66	0.75	0.37
32	3.17	0.94	0.91	1.41	0.60
64	6.23	1.81	1.78	2.95	1.23
128	15.17	5.14	5.03	6.34	3.33

All times in seconds. Cell shading encodes each primitive’s share of total accounted time per row. Linear layers execute locally as GPU-native GEMMs on additive shares (no interaction). Softmax grows as $\approx O(n^{1.5})$; Beaver and SiLU as $\approx O(n)$.

score matrix and therefore introduces a steeper (approximately quadratic) component as n grows. Crucially, SPEC executes public-weight linear layers locally as GPU-native GEMMs on additive shares, so end-to-end TTFT is driven by a small set of interactive kernels rather than by the linear compute; §5.3 quantifies where time and bytes concentrate.

Comparison to prior work. Because threat models (2PC vs. 3PC), model variants, and hardware/network settings differ across systems, Table 2 is not a controlled head-to-head benchmark; it is intended to situate SPEC’s latency/communication regime. At short prompts ($n=4, 8$), SPEC reduces TTFT by **one to three orders of magnitude** relative to prior work while keeping online communication in the same range (or lower for several baselines). Notably, NEXUS reports *lower* online communication than SPEC yet remains substantially *slower* in TTFT, indicating that end-to-end latency is not explained by bandwidth alone (e.g., interaction structure/round trips and GPU kernel efficiency can dominate). At a more realistic prompt length ($n=128$), SPEC achieves **9.78 s** TTFT while exchanging **4.66 GB** online in our intra-server two-GPU setting; §5.4 then isolates how TTFT degrades under degraded bandwidth/RTT.

5.3 Breakdown: Time and Communication by Primitive

SPEC’s end-to-end TTFT is dominated by a small number of *interactive* kernels. Public-weight linear layers run locally as GPU-native GEMMs on additive shares and therefore do not appear as online bottlenecks; the remaining cost concentrates in the secure non-linearities and secret–secret products.

Table 4: Online time/communication breakdown by primitive for $n = 128$ (sent+received communication).

Primitive	Time (%)	Comm (GiB)	Comm (%)
SiLU	35.4	1.93	41.4
Softmax	37.3	0.96	20.7
Beaver	25.0	1.77	37.9
Recon.	0.2	≈ 0	≈ 0

Table 4 reports the primitive-level attribution for a representative longer prompt ($n=128$), which we use as a concrete optimization case study.

Time vs. Bytes Are Not Proportional. At $n=128$, Softmax accounts for a larger fraction of time than of bytes exchanged, while SiLU and Beaver-based products account for a larger fraction of bytes than of time (Table 4). This gap indicates that improving TTFT is not only a matter of reducing total communication: some kernels are more *latency/round-trip* sensitive (time-heavy, byte-light), while others are more *bandwidth* sensitive (byte-heavy).

Implication. Future TTFT improvements should therefore focus on these few primitives: reduce round trips and improve overlap for Softmax-like kernels, and reduce message volume (e.g., packing/quantization/compression) for activation- and matmul-sized kernels such as SiLU and Beaver products. Reconstruction is negligible.

5.4 Sensitivity to Network Conditions

SPEC’s online cost is concentrated in a few interactive primitives (§5.3), so end-to-end TTFT can become network-bound when the two servers are not co-located. We therefore measure how bandwidth, RTT, jitter, and packet loss affect SPEC’s *online* TTFT.

Methodology. We emulate network conditions using Linux `tc netem`. To ensure the shaping affects communication even when both parties run on the same machine, we force the runtime to use TCP sockets (NCCL socket transport over loopback) rather than NCCL’s shared-memory/GPU-direct fast path. We report TTFT for a fixed prompt length ($n=16$, batch size 1), excluding offline preprocessing.

Overall Sensitivity. Table 5 shows two distinct regimes. In the *bandwidth-limited* regime (sub-millisecond RTT), TTFT degrades modestly down to 1 Gbps (1.9 $\times$) but rises sharply at 100 Mbps (11.0 $\times$).

Table 5: 2PC inference sensitivity to network conditions (end-to-end + per-operation breakdown). All times in seconds.

Category	Scenario	BW	RTT	SiLU (s)	Softmax (s)	Beaver (s)	Recon. (s)	E2E (s)	Slowdown
Baseline	GPU Direct	–	~0	0.8	0.4	0.8	0.0	2.2	–
<i>Bandwidth sweep (RTT = 0.1 ms)</i>									
	10 Gbps	10G	0.1 ms	0.8	0.5	0.8	0.0	2.4	1.1×
	1 Gbps	1G	0.1 ms	1.9	0.5	1.5	0.1	4.2	1.9×
	100 Mbps	100M	0.1 ms	13.4	1.0	9.8	0.1	24.5	11.0×
<i>Latency sweep (BW = 10 Gbps)</i>									
	1 ms	10G	1 ms	1.1	1.4	1.1	0.2	4.0	1.8×
	5 ms	10G	5 ms	2.1	5.6	2.7	0.7	11.4	5.1×
	50 ms	10G	50 ms	20.5	51.7	20.4	6.7	99.5	44.6×
<i>Network quality (BW = 10 Gbps)</i>									
	1 ms + 0.1% loss	10G	1 ms	1.1	1.5	1.4	0.2	4.3	1.9×
	1 ms + 1.0% loss	10G	1 ms	1.8	2.3	3.6	0.2	8.1	3.6×
	5 ms ± 1 ms	10G	5 ± 1 ms	2.4	5.9	2.9	0.8	12.2	5.5×
	5 ms ± 5 ms	10G	5 ± 5 ms	4.4	7.4	4.2	1.0	17.2	7.7×

1× 2× 5× 15× 30× slowdown

Simulated via Linux `tc netem` over full-duplex Ethernet; E2E denotes 3-token generation. Jitter is drawn from a normal distribution. Cell shading encodes per-operation slowdown relative to the GPU-Direct baseline. 1% packet loss inflates latency through TCP retransmission overhead. All experiments use TCP sockets (NCCL socket transport); RDMA-capable interconnects (e.g., InfiniBand, RoCE) would reduce both latency and bandwidth sensitivity, but cannot be faithfully emulated under `tc netem` shaping.

In the *latency-limited* regime (10 Gbps), TTFT grows rapidly with RTT, reaching 44.6× at 50 ms—indicating that round-trip delays dominate once RTT is large.

Which Primitives Drive Slowdown. Table 5 attributes the slowdown to specific primitives. At low RTT but constrained bandwidth (100 Mbps), SiLU and Beaver matmul dominate TTFT (13.4 s and 9.8 s out of 24.5 s), consistent with their large activation-/matmul-sized message payloads. At high RTT (50 ms), Softmax becomes the primary bottleneck (51.7 s, ≈52% of E2E), consistent with substantially more sequential exchanges (round trips) than the other primitives.

Implications. These results suggest two complementary optimization levers for real deployments: (1) reduce or better-overlap round trips in Softmax-like kernels to tolerate higher RTT, and (2) reduce message volume for bandwidth-heavy kernels (SiLU and Beaver products) via packing and quantization/compression. Packet loss further amplifies these costs via retransmissions (Table 5).

6 Conclusion

We presented SPEC, a two-server secure inference system for transformer LLMs in the honest-but-curious,

non-colluding servers model. SPEC exploits additive secret sharing to execute the dominant linear components of transformer inference locally as GPU-native floating-point GEMMs on shares, while securely evaluating only the unavoidable interaction points using a hybrid of Beaver-triple products and FSS-based non-linear protocols. By moving secure computation from “everything” to a small set of well-defined kernels, SPEC makes modern secure LLM inference substantially more practical.

On Llama-3.1-8B-Instruct, SPEC achieves end-to-end time-to-first-token (TTFT) of 1.33 s at $n=8$ and 9.78 s at $n=128$, with online communication of 0.22 GB and 4.66 GB respectively. Compared to recent secure transformer/LLM inference systems, SPEC reduces TTFT by one to two orders of magnitude at short prompts while keeping online communication competitive. Our breakdown further shows that online cost concentrates in a few interactive primitives (Softmax, activations, and secret-secret products), providing a clear roadmap for continued optimization.

Overall, SPEC closes much of the gap between cryptographic privacy and real-time transformer inference by aligning the protocol boundary with the transformer’s algebraic structure and GPU hardware realities.

Ethical Considerations

This work proposes a privacy-preserving inference protocol based on multi-party computation, with no reliance on trusted hardware. Our system ensures that both model parameters and user inputs remain secret-shared throughout inference. All experiments are conducted using publicly available models and synthetically generated data. No PII, sensitive content, or human subjects are involved.

We adhere to the ethical research standards outlined by USENIX Security. Our implementation does not compromise user privacy or safety, and no real-world data collection or deployment was performed.

Potential Misuse. As with any cryptographic system, misuse is possible. In particular, malicious parties could exploit inference privacy to conceal malicious or abusive inputs from service providers. However, our protocol does not enable anonymity, untraceable queries, or deniable usage beyond existing baselines. We recommend responsible deployment practices, including audit logging and access control on the client side.

Availability

SPEC’s source code, configuration files, and benchmarking scripts are included in the submitted artifact and will be released publicly upon acceptance.

References

- [1] Yoshimasa Akimoto, Kazuto Fukuchi, Youhei Akimoto, and Jun Sakuma. Privformer: Privacy-preserving transformer with mpc. *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*, null:392–410, 2023.
- [2] Anonymous et al. THOR: Privacy-preserving transformer inference with homomorphic encryption. *Cryptology ePrint Archive*, Paper 2024/1881, 2024.
- [3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. Layer normalization. In *arXiv*, 2016.
- [4] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO ’91*, volume 576 of *Lecture Notes in Computer Science*, pages 420–432. Springer Berlin Heidelberg, 1992.
- [5] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC)*, pages 1–10, 1988.
- [6] Elette Boyle, Nishanth Chandran, N. Gilboa, Divya Gupta, Yuval Ishai, Nishant Kumar, and Mayank Rathee. Function secret sharing for mixed-mode and fixed-point secure computation. *IACR Cryptol. ePrint Arch.*, 2020:1392, 2020.
- [7] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function secret sharing. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015*, pages 337–367, Berlin, Heidelberg, 2015. Springer Berlin Heidelberg.
- [8] Elette Boyle, Niv Gilboa, and Yuval Ishai. Function Secret Sharing: Improvements and Extensions. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1292–1303, Vienna Austria, October 2016. ACM.
- [9] Elette Boyle, Niv Gilboa, and Yuval Ishai. Secure computation with preprocessing via function secret sharing. *Cryptology ePrint Archive*, Paper 2019/1095, 2019.
- [10] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In *27th USENIX Security Symposium (USENIX Security 18)*, 2018.
- [11] Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, 2000.
- [12] Ronald Cramer, Ivan Bjerre Damgård, and Jesper Buus Nielsen. *Secure multiparty computation*. Cambridge University Press, 2015.
- [13] Ye Dong, Wen jie Lu, Yancheng Zheng, Haoqi Wu, Derun Zhao, Jin Tan, Zhicong Huang, Cheng Hong, Tao Wei, and Wenguang Chen. PUMA: Secure inference of LLaMA-7b in five minutes. *Security and Safety*, 4:2025014, 2025.
- [14] Ye Dong, Wen-Jie Lu, Yancheng Zheng, Haoqi Wu, Derun Zhao, Jin Tan, Zhicong Huang, Cheng Hong, Tao Wei, and Wenguang Chen. Puma: Secure inference of llama-7b in five minutes. *arXiv preprint arXiv:2307.12533*, 2023.
- [15] Stefan Elfving, Eiji Uchibe, and Kenji Doya. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. In *Neural Networks*, 2018.

- [16] David Evans, Vladimir Kolesnikov, and Mike Rosulek. A pragmatic introduction to secure multi-party computation. *Foundations and Trends in Privacy and Security*, 2(2-3):70–246, 2018.
- [17] Oded Goldreich. *Foundations of Cryptography*. Cambridge University Press, 1 edition, May 2004.
- [18] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.
- [19] Kanav Gupta, Nishanth Chandran, Divya Gupta, Jonathan Katz, and Rahul Sharma. Shark: Actively secure inference using function secret sharing. Cryptology ePrint Archive, Paper 2025/716, 2025.
- [20] Kanav Gupta, Nishanth Chandran, Divya Gupta, Jonathan Katz, and Rahul Sharma. Shark: Actively secure inference using function secret sharing. *2025 IEEE Symposium on Security and Privacy (SP)*, null:2472–2490, 2025.
- [21] Kanav Gupta, Neha Jawalkar, Ananta Mukherjee, Nishanth Chandran, Divya Gupta, A. Panwar, and Rahul Sharma. Sigma: Secure gpt inference with function secret sharing. *Proc. Priv. Enhancing Technol.*, 2024:61–79, 2024.
- [22] Kanav Gupta, Neha Jawalkar, Ananta Mukherjee, Nishanth Chandran, Divya Gupta, Ashish Panwar, and Rahul Sharma. SIGMA: Secure GPT inference with function secret sharing. *Proceedings on Privacy Enhancing Technologies*, 2024(4):61–79, 2024.
- [23] Meng Hao, Hongwei Li, Hanxiao Chen, Pengzhi Xing, Guowen Xu, and Tianwei Zhang. Iron: Private inference on transformers. In *Proceedings of the 36th International Conference on Neural Information Processing Systems, NIPS ’22*, Red Hook, NY, USA, 2022.
- [24] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (GELUs). In *arXiv*, 2016.
- [25] Aylin Ileri, Seungwan Kim, Assia Tria, Nhi Ho, Mark Ibrahim, Qianxiao Li, Marcos Silva, Ce Zhang, and David J. Wu. ENSI: Non-interactive secure inference for large language models. *arXiv*, 2025.
- [26] Patrick Jauernig, Ahmad-Reza Sadeghi, and Emmanuel Stappf. Trusted execution environments: Properties, applications, and challenges. *IEEE Security & Privacy*, 18(2):56–60, 2020.
- [27] Wenjie Lu, Zhicong Huang, Zhen Gu, Jingyu Li, Jian Liu, Cheng Hong, Kui Ren, Tao Wei, and Wenguang Chen. BumbleBee: Secure two-party inference framework for large transformers. In *Proceedings 2025 Network and Distributed System Security Symposium*, San Diego, CA, USA, 2025.
- [28] Peter Kairouz et al. Advances and open problems in federated learning. *Foundations and Trends in Machine Learning*, 14(1-2):1–210, 2021.
- [29] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. Spectre attacks: Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy (SP)*, 2019.
- [30] Ryan T. Lehmkuhl, Pratyush Mishra, Akshayaram Srinivasan, and Raluca A. Popa. Muse: Secure inference resilient to malicious clients. *IACR Cryptol. ePrint Arch.*, 2021:1040, 2021.
- [31] Dacheng Li, Hongyi Wang, Rulin Shao, Han Guo, Eric Xing, and Hao Zhang. MPCFORMER: Fast, performant and private transformer inference with MPC. In *The Eleventh International Conference on Learning Representations*, 2023.
- [32] Haoran Li, Mingshi Xu, and Yangqiu Song. Sentence embedding leaks more information than you expect: Generative embedding inversion attack to recover the whole sentence. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 14022–14040, Toronto, Canada, 2023.
- [33] Yehuda Lindell. How to simulate it - a tutorial on the simulation proof technique. Cryptology ePrint Archive, Paper 2016/046, 2016.
- [34] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security 18)*, 2018.
- [35] Fengrun Liu, Xiang Xie, and Yu Yu. Scalable multi-party computation protocols for machine learning in the honest-majority setting. 2024.
- [36] Jing Liu, Jamie Cui, and Cen Chen. Online efficient secure logistic regression based on function secret sharing. *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, null:null, 2023.

- [37] Llama.cpp-MPC Authors. Llama.cpp-MPC: An MPC-enabled implementation of llm inference based on llama.cpp. Technical report / open-source project, 2024.
- [38] Chuan Ma, Jun Li, Kang Wei, Bo Liu, Ming Ding, Long Yuan, Zhu Han, and H Vincent Poor. Trusted ai in multi-agent systems: An overview of privacy and security for distributed learning. *arXiv preprint arXiv:2202.09027*, 2022.
- [39] Peihua Mai, Ran Yan, Zhe Huang, Youjia Yang, and Yan Pang. Split-and-denoise: Protect large language model inference with local differential privacy. *arXiv*, 2023.
- [40] NVIDIA. Confidential computing on NVIDIA Hopper H100. Whitepaper / technical report, 2024.
- [41] 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)*, pages 4753–4771, San Francisco, CA, USA, 2024.
- [42] Nicolas Papernot, Patrick McDaniel, Arunesh Sinha, and Michael Wellman. Towards the science of security and privacy in machine learning. *arXiv preprint arXiv:1611.03814*, 2016.
- [43] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions. In *arXiv*, 2017.
- [44] Deevashwer Rathee, Anwesh Bhattacharya, Divya Gupta, Rahul Sharma, and D. Song. Secure floating-point training. 2023.
- [45] Deevashwer Rathee, Dacheng Li, Ion Stoica, Hao Zhang, and Raluca Ada Popa. MPC-minimized secure LLM inference, 2025.
- [46] Nicola Rieke, Jeremy Hancox, Wenqi Li, Fausto Milletari, Holger R. Roth, Shadi Albarqouni, Spyridon Bakas, Mathieu N. Galtier, Bennett A. Landman, Klaus Maier-Hein, Sebastien Ourselin, Micah Sheller, Ronald M. Summers, Andrew Trask, Daguang Xu, Maximilian Baust, M. Jorge Cardoso, F. H. Peterson, et al. The future of digital health with federated learning. *npj Digital Medicine*, 3(1):119, 2020.
- [47] Manuel B. Santos, Dimitris Mouris, Mehmet Ugurbil, Stanislaw Jarecki, Jos’e Reis, Shubho Sengupta, and Miguel de Vega. Curl: Private llms through wavelet-encoded look-up tables. 2024.
- [48] Kyle Storrier, Adithya Vadapalli, Allan Lyons, and Ryan Henry. Grotto: Screaming fast (2+1)-pc or $\mathbb{Z}_2$ via (2,2)-dpfs. *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, null:null, 2023.
- [49] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. Stealing machine learning models via prediction APIs. In *25th USENIX Security Symposium (USENIX Security 16)*, Austin, TX, 2016.
- [50] Mehmet Ugurbil, Dimitris Mouris, Manuel B. Santos, José Cabrero-Holgueras, Miguel de Vega, and Shubho Sengupta. Fission: Distributed privacy-preserving large language model inference. *Cryptology ePrint Archive*, Paper 2025/653, 2025.
- [51] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, 2017.
- [52] Sameer Wagh. Barnowl: Secure comparisons using silent pseudorandom correlation generators. *IACR Cryptol. ePrint Arch.*, 2022:800, 2022.
- [53] Sameer Wagh. Pika: Secure computation using function secret sharing over rings. *Proc. Priv. Enhancing Technol.*, 2022:351–377, 2022.
- [54] Haoqi Wu, Wenjing Fang, Yancheng Zheng, Junming Ma, Jin Tan, and Lei Wang. Ditto: Quantization-aware secure inference of transformers upon MPC. In *Proceedings of the 41st International Conference on Machine Learning, ICML’24*, 2024.
- [55] 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. *Cryptology ePrint Archive*, Paper 2025/1532, 2025.
- [56] Peng Yang, Z. L. Jiang, Shiqi Gao, Jiehang Zhuang, Hongxiao Wang, Jun bin Fang, S. Yiu, and Yulin Wu. Fssnn: Communication-efficient secure neural network training via function secret sharing. *IACR Cryptol. ePrint Arch.*, 2023:73, 2023.
- [57] Peng Yang, Zoe Lin Jiang, Shiqi Gao, Hongxiao Wang, Jun Zhou, Yangyiye Jin, Siu-Ming Yiu, and Junbin Fang. FssNN: Communication-efficient secure neural network training via function secret sharing. *Cryptology ePrint Archive*, Paper 2023/073, 2023.

- [58] Andrew C. Yao. Protocols for secure computations. In *23rd Annual Symposium on Foundations of Computer Science (FOCS)*, pages 160–164, 1982.
- [59] Zhichao You, Xuewen Dong, Ke Cheng, Xutong Mu, Jiaxuan Fu, Shiyang Ma, Qiang Qu, and Yulong Shen. Prifft: Privacy-preserving federated fine-tuning of large language models via hybrid secret sharing. *arXiv*, 2025.
- [60] Han-Qing Zeng, Zhe Xia, Hanyu Yang, Wentao Wu, Huili Chen, Jie Chen, Jian Liu, Kui Ren, and Wen jie Lu. MPCache: Fast private LLM inference with MPC-friendly KV caching. *arXiv*, 2025.
- [61] Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems*, 2019.
- [62] Dongdong Zhang, Wei Xiao, Yingtong Wang, Mingliang Yu, and Gang Wang. Privacy-preserving machine learning: Cryptographic techniques and development frontiers. *2025 4th International Symposium on Computer Applications and Information Technology (ISCAIT)*, null:1788–1793, 2025.
- [63] 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 *Proceedings 2025 Network and Distributed System Security Symposium*, San Diego, CA, USA, 2025.