Hyperion: Private Token Sampling with Homomorphic Encryption

Lawrence Lim[1], Jiaming Liu[1], Vikas Kalagi[1], Divyakant Agrawal[1], Amr El Abbadi[1]

[1]University of California, Santa Barbara

Correspondence: lawrenceklim@ucsb.edu

Abstract

A promising direction for enabling private queries to large language models (LLMs) is with homomorphic encryption (HE). An open problem is performing token sampling under HE. In this paper, we introduce Hyperion, an efficient HE algorithm for inverse transform sampling, enabling private token sampling with 1 comparison depth, $O(1)$ amortized comparisons, and $O(\log n)$ rotations. We implement our approach and demonstrate that it samples tokens in 0.14 seconds for 32k tokens ($\approx 4.4\,\mu$s per token) on GPU, achieving a $100\times$ latency improvement over prior work.

1 Introduction

Large Language Models (LLMs) enable an extraordinary range of applications, yet their broader adoption is often limited by a fundamental concern: users must send sensitive data to remote model providers. Medical practitioners may want to use LLMs to summarize patient history, yet querying LLMs may expose sensitive data of their patients. Many individuals and organizations may want the capabilities of LLMs, but are unwilling to compromise the privacy of their inputs.

Homomorphic encryption (HE) has emerged as a promising foundation for private LLM inference by enabling computations on ciphertexts. A user can encrypt their query and send it to the model provider. The model provider can perform the same matrix multiplications, nonlinear activations, and token-generation steps as in standard LLM inference directly on the ciphertext while learning nothing about the underlying query. The provider then returns an encrypted result that only the user can decrypt.

While recent systems have demonstrated the feasibility of homomorphic LLM inference (Moon et al., 2025; Zhang et al., 2024; Park et al., 2025; De Castro et al., 2025; Rovida and Leporati, 2024; Rho et al., 2025a), a key challenge remains unresolved: how to privately and efficiently perform token sampling during generation. A straightforward approach requires interactive communication between the user and provider to sample a token from encrypted logits. Because this results in communication latency for each token, a better solution may be to sample privately without requiring communication (i.e. non-interactively). However, existing non-interactive solutions (e.g., NEXUS (Zhang et al., 2024), EncryptedLLM (De Castro et al., 2025)) instead rely on homomorphic ARGMAX to deterministically select the highest-probability token. This is undesirable for two reasons. First, LLM token sampling, in practice, is inherently randomized, and deterministic ARGMAX cannot replicate behaviors such as temperature sampling. Second, homomorphic ARGMAX is computationally expensive, often requiring either $O(\log n)$ comparison depth (Zhang et al., 2024) or $O(n^2)$ comparisons (Mazzone et al., 2025; Kim et al., 2025) to identify the maximum among n vocabulary elements. As a concrete example, the authors of EncryptedLLM (De Castro et al., 2025) report that homomorphic ARGMAX accounts for approximately 8 seconds ($\approx 12\%$) of a total forward pass time of 68 seconds.

In this paper, we present **Hyperion**, an efficient HE algorithm for randomized token sampling that addresses the limitations of homomorphic ARGMAX. Hyperion homomorphically enables inverse transform sampling, a classical algorithm for drawing samples from an arbitrary discrete distribution. The resulting HE construction is efficient, requiring only $O(\log n)$ rotations and only $O(1)$ amortized comparisons and 1 comparison depth. We implemented Hyperion on GPU, demonstrating its low latency ($4.4\,\mu$s per token) and a $100\times$ improvement over baselines.

Several prior works have explored private LLM inference with HE (Moon et al., 2025; Zhang et al., 2024; Park et al., 2025; De Castro et al., 2025;

Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 14150–14159
July 2-7, 2026 ©2026 Association for Computational Linguistics

Rovida and Leporati, 2024; Rho et al., 2025a; Zhang et al., 2025). These works primarily focus on the transformer model in HE, with NEXUS (Zhang et al., 2024) also describing a method for private token generation through a private ARGMAX. This approach is undesirable because of its determinism and computational expense. More recent concurrent work investigates randomized private token sampling rather than deterministic selection (Avitan et al., 2025; Rho et al., 2025b). Avitan *et al.* introduces a CUTMAX approach for nucleus sampling (Avitan et al., 2025). Meanwhile, Rho *et al.* enhances stability of an inverse transform sampling approach by reordering tokens (Rho et al., 2025b). In contrast, we investigate the accuracy of the SIGN approximation that underpins the inverse transform sampling, and our proposed algorithm, Hyperion, achieves significantly better performance.

2 Preliminaries

2.1 Large Language Model Text Generation

A large language model (LLM) is a neural network that typically consists of an embedding layer followed by a number of transformer layers, ending with an output layer that produces a vector of logits. Each logit corresponds to a token in the vocabulary. These logits are unnormalized scores indicating how likely each token is to be generated next. To sample a token from this vector of logits, a softmax with temperature T is applied, converting the logits into a probability distribution over the vocabulary. More formally, for a vector $x \in \mathbb{R}^n$, the softmax probability assigned to index i is

$$\text{softmax}(x)_i = \frac{e^{x_i/T}}{\sum_{j=1}^{n} e^{x_j/T}}. \tag{1}$$

The temperature parameter controls the *sharpness* of the distribution. A smaller temperature T (i.e. closer to 0) makes high-logit tokens more dominant, produces more deterministic outputs while a larger T flattens the distribution, increasing randomness in the sampled token. Sampling from this distribution yields the next token, which is then fed back into the model to generate subsequent tokens.

To sample from the distribution, a known method to do so is called *inverse transform sampling* (Wu et al., 2001). Developing a privacy-preserving method of sampling a token constitutes the primary contribution of this work. We discuss this in detail in Section 3.

2.2 Homomorphic Encryption

Homomorphic encryption enables computation on encrypted data, enabling applications such as private LLM inference. We use CKKS (Cheon et al., 2017), an approximate homomorphic encryption scheme in which both plaintexts and ciphertexts encode vectors of floating point numbers. Homomorphic operations act element-wise on these vectors in a SIMD manner. The vector length, or number of slots, is $s = N/2$, where the ring dimension is a power of two $N = 2^k$ (commonly $N = 2^{16}$). Besides key generation, encryption, and decryption, CKKS supports several homomorphic operations:

- ADD: Element-wise addition of two ciphertexts or a ciphertext and a plaintext.
- MULTIPLY: Element-wise multiplication of two ciphertexts or a ciphertext and a plaintext.
- ROTATE: A cyclic left shift of the vector slots. We denote $\mathsf{Rotate}(c, i)$ as the rotation of ciphertext c by i positions.

Efficient use of CKKS requires minimizing both multiplicative depth and key-switching operations. Each ciphertext carries a limited multiplicative depth budget that decreases with every multiplication. When this budget runs out, an extremely expensive procedure called *bootstrapping* is required to refresh the ciphertext's budget. Thus, reducing multiplicative depth (and therefore bootstrapping) is crucial for performance. The next most costly operations are key-switching operations, which occur during ciphertext–ciphertext multiplications and rotations, and should likewise be minimized.

Prior work has developed privacy-preserving LLM systems using HE (Zhang et al., 2024; Moon et al., 2025; Park et al., 2025; Rho et al., 2025a; De Castro et al., 2025; Zhang et al., 2025; Rovida and Leporati, 2024). These systems allow the user to encrypt their input, and the model provider to evaluate the LLM over the encrypted input. These works enable private inference by translating every operation in the transformer model, such as nonlinear operations and matrix multiplications, into the CKKS operations of ADD, MULTIPLY, and ROTATE.

Prior systems, such as NEXUS (Zhang et al., 2024), have used ARGMAX to sample a token. This is expensive because they require $O(\log n)$ depth comparisons via a polynomial SIGN approximation, resulting in $O(\log n)$ bootstrapping operations. An

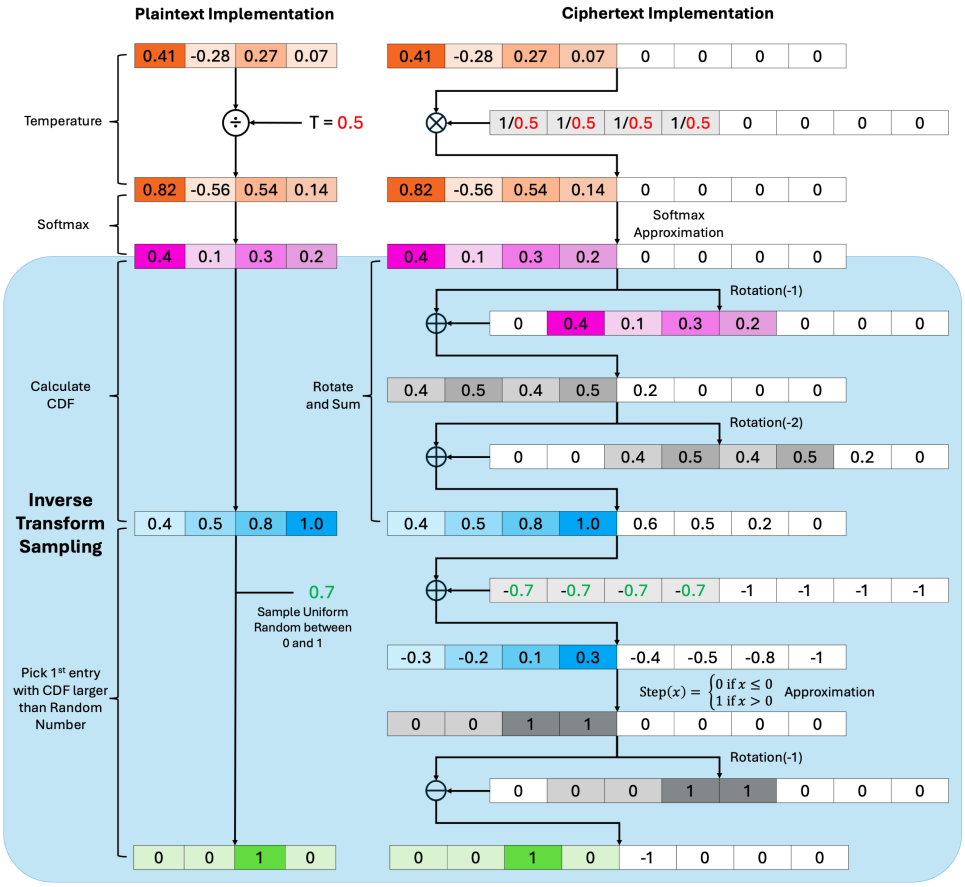


Figure 1: An illustration of Hyperion enabling the privately sampling of a token from logits.

$O(1)$ depth method of calculating the max is via rank sorting (Mazzone et al., 2025; Kim et al., 2025). Rank sorting requires expensive $O(n^2)$ SIGN approximations, incurring many ciphertext-ciphertext multiplications during the SIGN approximation and are unable to efficiently scale to typical vocabulary sizes of 32,000. Regardless, the determinism of max makes them a poor choice for token sampling in private LLMs.

3 Private Token Sampling

Our goal is to enable efficient private token sampling. To this end, we develop an efficient homomorphic encryption method for *inverse transform sampling*, a standard technique for sampling from an arbitrary distribution. Figure 1 provides a high-level overview of this procedure: the left side depicts a plaintext implementation of the algorithm, while the right side shows its encrypted counterpart. As in conventional token sampling from logits, the input logits are first transformed by a temperature-scaled softmax to obtain a probability mass function. The inverse transform sampling algorithm begins with converting this probability mass function into its cumulative distribution function (CDF). Next, a uniform random number on $[0, 1]$ is drawn, and the sampled token is taken to be the first index whose CDF exceeds this value. The selected token is finally represented as a one-hot encoded vector.

Our homomorphic encryption method for token sampling proceeds as follows. We begin by assuming that half of the slots in the ciphertext are zero-padded, a requirement that enables efficient computation of the CDF. We discuss how to support a larger vocabulary than half of slots later. We apply an approximation of a temperature-scaled softmax. Here, we rely on prior work which has proposed various softmax approximations under homomorphic encryption (Cho et al., 2024; Moon et al., 2025; Lim et al., 2025; Zhang et al., 2024). To compute the CDF needed for inverse transform sampling, we use a rotate-and-sum procedure (Algorithm 1, ROTATEANDSUM) that performs right rotations by powers of two followed by additions. This requires only $O(\log n)$ rotations and additions and no multiplications, making the procedure both efficient and highly scalable. Next, a uniform random number is sampled and subtracted from every CDF entry.

Algorithm 1 Rotate-and-Sum

Computes a prefix sum or cumulative distribution function (CDF). Ciphertext c encrypts n values with zero padding on at least half of all SIMD slots.

def RotateAndSum(c, n)
  $\ell \leftarrow \lceil \log_2(n) \rceil$
  $\mathsf{sum} \leftarrow c$
  for $i = 0$ to $\ell - 1$ **do**
    $\mathsf{sum} \leftarrow \mathsf{sum} + \mathsf{Rotate}(\mathsf{sum}, -2^i)$
  return sum

Computes a prefix sum or cumulative distribution function (CDF), over the sequence of ciphertexts $\{c_i\}_{i=1}^k$, where each ciphertext encrypts $\{n_i\}_{i=1}^k$ values. The first $k-1$ ciphertexts encrypt values in half the available slots $n_1 = n_2 = \cdots = n_{k-1} = N/4 = s/2$ and the remaining slots are zero-padded.

def MultiRotateAndSum($\{c_i, n_i\}_{i=1}^k$)
  $\mathsf{ret} \leftarrow [\,]$
  for $i = 1$ to k **do**
    $\mathsf{cdf} \leftarrow$ RotateAndSum(c_i, n_i)
    if $i = 1$ **then**
      $\mathsf{ret}_i \leftarrow \mathsf{cdf}$
    else
      $\mathsf{ret}_i \leftarrow \mathsf{cdf} + \mathsf{sum}$
    if $i + 1 < k$ **then**
      $\mathsf{sum} \leftarrow \mathsf{ret}_i + \mathsf{Rotate}(\mathsf{cdf}, 2^{\lceil \log_2(n_i) \rceil})$
  return ret

This produces a vector that consists of zero or more negative values followed by positive values; the first positive entry corresponds to the index whose CDF exceeds the sampled random value. To detect this index homomorphically, we apply a unit step function approximation (also known as a Heaviside function) mapping negative values to 0 and positive values to 1. The unit step function approximation is a modification of a Sign approximation, where $\text{Step}(x) = \frac{1}{2} \cdot \text{Sign}(x) + 1$ and the Sign function is defined as follows:

$$\text{Sign}(x) = \begin{cases} -1 & \text{if } x < 0, \\ 0 & \text{if } x = 0, \\ 1 & \text{if } x > 0 \end{cases}$$

Assuming a good Sign approximation (which we discuss in Section 4), this results in a vector where there are zero or more 0 values, then a series of 1

Algorithm 2 Combine Ciphertexts

Combines pairs of ciphertexts to use available slots s more efficiently before the Sign approximation. Takes in a sequence of ciphertexts $\{c_i\}_{i=1}^k$ where the first $k-1$ ciphertexts encrypt CDF values in the first half of slots $n_1 = n_2 = \cdots = n_{k-1} = N/4 = s/2$.

def CombineCtxts($\{c_i, n_i\}_{i=1}^k$)
  for $i = 1$ to k **do** ▷ First mask ciphertext.
    $c_i \leftarrow c_i \times (\underbrace{1, \ldots, 1}_{n_i}, \underbrace{0, \ldots, 0}_{s-n_i})$
  $\mathsf{ret} \leftarrow [\,]$
  for $i = 1$ to $\lfloor k/2 \rfloor$ **do**
    $\mathsf{ret}_i \leftarrow c_{2i-1} + \mathsf{Rotate}(c_{2i}, -s/2)$
  if $k \mod 2 = 1$ **then**
    $\mathsf{ret}_{\lceil k/2 \rceil} \leftarrow c_k$
  return ret

values. Finally, we isolate the one-hot encoding of the selected token by performing a single right rotation of this vector and subtracting it from the original.

Supporting larger vocabulary sizes: When the vocabulary size exceeds half of the available ciphertext slots, Hyperion remains correct by distributing the logits across multiple ciphertexts. For a typical ring dimension of $N = 2^{16}$, zero-padding leaves $N/4 = 2^{14}$ usable slots per ciphertext, while LLaMA-1 and LLaMA-2 have vocabularies of roughly 32,000 tokens. Thus, two ciphertexts, with half of their slots zero-padded are sufficient to represent the logits. The CDF can still be computed efficiently in this regime. In Algorithm 1, MultiRotateAndSum computes the CDF by applying RotateAndSum independently to each ciphertext, followed by an additional left rotation to propagate the running sum across ciphertexts.

Optimizing slot use with CombineCtxts: We then apply an optimization (Algorithm 2) that improves performance by fully utilizing all available slots s, rather than only $s/2$, during the Sign (or Step) approximation. Since the Sign approximation dominates Hyperion's runtime, we invoke CombineCtxts immediately after MultiRotateAndSum to maximize slot utilization before evaluating the Sign polynomial. This optimization trades one additional level of multiplicative depth for an

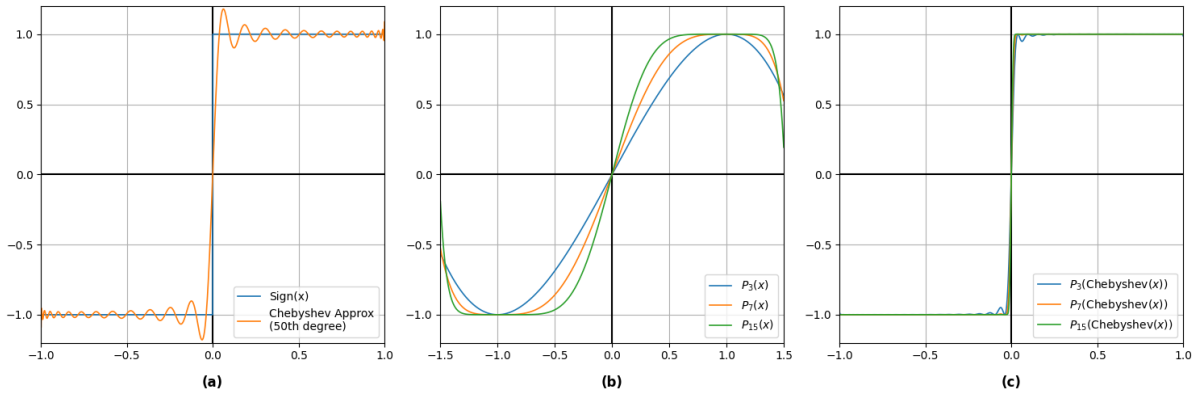


Figure 2: (a) A 50th degree Chebyshev approximation of the SIGN function. (b) Three smoothing polynomials with $P_3 = \frac{3}{2}x - \frac{1}{2}x^3$, $P_7(x) = \frac{35}{16}x - \frac{35}{16}x^3 + \frac{21}{16}x^5 - \frac{5}{16}x^7$, and $P_{15}(x) = \frac{6435}{2048}x - \frac{15015}{2048}x^3 + \frac{27027}{2048}x^5 - \frac{32175}{2048}x^7 + \frac{25025}{2048}x^9 - \frac{12285}{2048}x^{11} + \frac{3465}{2048}x^{13} - \frac{429}{2048}x^{15}$. (c) Composition of the Chebyshev approximation with the smoothing polynomials.

approximately $2\times$ reduction in SIGN cost.

After that, a uniform random value is sampled and subtracted from each ciphertext, after which a SIGN (or unit Step) approximation is applied independently to each ciphertext. To obtain a final one-hot encoding, the rightmost slot of the first ciphertext is transferred to the second ciphertext using a rotation followed by a masking operation. Finally, Hyperion precomputes all plaintexts required (e.g., for masking operations in COMBINECTXTS), since these values are known *a priori*, further reducing end-to-end runtime overhead.

Overall, our private inverse transform sampling requires only $O(1)$ amortized comparisons and comparison depth (arising from the step-function approximation), and incurs at most two additional multiplicative levels when multiple ciphertexts are involved (one for COMBINECTXTS and another for the final right-rotation step). Lastly, the rotate-and-sum procedure requires only $O(\log n)$ rotations, while obtaining a final one-hot encoding uses only $O(1)$ additional rotations.

Difference from concurrent work: Rho *et al.* (Rho et al., 2025b) also propose a private implementation of the inverse transform sampling, but their design differs from Hyperion in several important ways. First, their primary focus is on improving stability by reordering tokens via a traveling-salesman–style heuristic, whereas we additionally investigate the accuracy of the SIGN approximation, which is presented in the following section. There are also key algorithmic differences. Specifically, Rho *et al.* compute the CDF through an encrypted matrix-multiplication routine, which incurs an additional multiplicative depth and often requires $O(\sqrt{n})$ rotations and $O(n)$ multiplications, even with the more optimized matrix-multiplication algorithms (Bossuat et al., 2021). In contrast, the CDF in Hyperion uses a ROTATEANDSUM procedure which only requires $O(\log n)$ rotations and no multiplications at this stage. Finally, their conversion to a one-hot encoding requires an additional ciphertext-ciphertext multiplication over Hyperion. As a result, Hyperion's overall inverse transform sampling is more efficient in multiplication and rotation complexity. In our evaluation (Section 5), we show that this results in around $10{,}000\times$ better performance.

4 SIGN Approximation

The correctness of Hyperion depends on the quality of the underlying SIGN approximation. If the approximation is poor, the resulting token sampling may be imperfect, and these small discrepancies can accumulate over multiple autoregressive steps in an LLM, compounding error across the generated sequence. Because SIGN is discontinuous, no polynomial approximation can reliably distinguish values extremely close to 0. Consequently, such approximations are only accurate over the range $[-1, -2^{-\epsilon}] \cup [2^{-\epsilon}, 1]$ for some ϵ. Even within this region, approximation errors can persist, so minimizing it is crucial.

Our goal is to minimize both categories of errors introduced by the SIGN approximation in private token sampling. To this end, we extend prior work (Cheon et al., 2019; Lee et al., 2022; Cheon et al., 2020) with a two-stage polynomial scheme. In the first stage, a coarse but near-correct polyno-

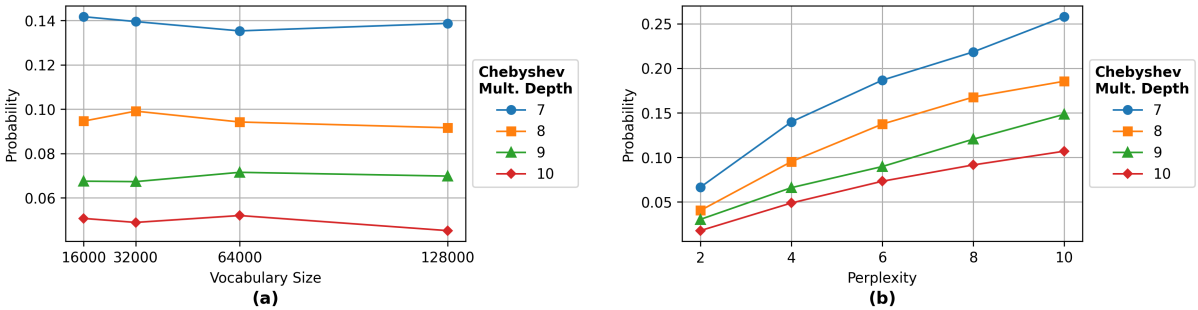


Figure 3: Probability that Hyperion evaluates the SIGN approximation within the indistinguishable region $[-2^{-\epsilon}, 2^{-\epsilon}]$, as defined in Table 1, for at least one input value. In (a), we vary the vocabulary size while fixing the perplexity to 4. In (b), we vary the perplexity while fixing the vocabulary size to 32,000.

mial rapidly pushes inputs toward either -1 or 1. We adopt Chebyshev approximations (Cheney, 1966) for this step because their steep slope near $x = 0$ sharply separates positive and negative inputs, thereby reducing the region around zero where the approximation is unable to distinguish SIGN. Alternative choices such as minimax polynomials and their compositions (Lee et al., 2022) may also serve effectively at this stage. The second polynomial then acts as a smoothing corrector for the residual error produced by the first stage; this is similar to the SIGN approximation of Cheon *et al.* (Cheon et al., 2020). Whereas Cheon *et al.* apply repeated compositions of lower-degree polynomials to drive convergence (Cheon et al., 2020), our use of a higher-degree endpoint-flat polynomial is specifically intended to suppress and smooth errors introduced in the first stage. We construct this smoothing polynomial to have degree $2n + 1$ and require the following:

- Exact interpolation at $(-1, -1)$ and $(1, 1)$
- The first through n-th derivatives are zero at both points.

These conditions ensure that the polynomial is maximally flat at the endpoints, so values already near ± 1 are preserved with high accuracy. As a result, the composed approximation closely matches the true SIGN function near its stable regions and reduces cumulative error during LLM token generation. These polynomials are derivable via Hermite interpolation (with the above constraints) as follows:

$$P_{2n+1}(x) = \frac{(2n+1)!}{2^{2n}(n!)^2} \sum_{k=0}^{n} \binom{n+k}{k} \frac{(-1)^k x^{2k+1}}{2k+1} \quad (2)$$

Figure 2 shows a SIGN approximation composed of both a Chebyshev approximation and smoothing polynomial. In panel (a), we plot a 50th-degree Chebyshev approximation of the SIGN function, showing the steep slope at $x = 0$ and the oscillatory deviations that remain. Panel (b) shows three smoothing polynomials, P_3, P_7, and P_{15}, whose degrees correspond to multiplicative depths of two, three, and four, respectively. Since each polynomial is constructed to be extremely flat at $x = \pm 1$, inputs already near ± 1 are mapped to values nearly indistinguishable from ± 1. For performance, Hyperion evaluates these polynomials using the Odd Baby-Step Giant-Step algorithm (Lee et al., 2022, 2020), which minimizes multiplicative depth and key-switching operations. Finally, panel (c) demonstrates the composition of these smoothing polynomials with the Chebyshev approximation: the residual oscillatory error is effectively suppressed, yielding an output profile that closely matches the ideal SIGN function across the domain.

Increasing the degree of the Chebyshev polynomial in our SIGN approximation improves its ability to distinguish values close to 0 (i.e. larger ϵ), but also increases both computational cost and multiplicative depth. Table 1 summarizes the resulting ϵ values when Chebyshev approximations of larger multiplicative depths are composed with the P_{15} polynomial (with 4 additional multiplicative depth).

Chebyshev Depth	7	8	9	10
ϵ	7.18	8.23	9.23	10.25

Table 1: Chebyshev Depth vs. ϵ where our SIGN approximation has a maximum error of 10^{-2}. ϵ can be interpreted as the number of bits of distinguishability.

A natural question is: *how accurate must the SIGN approximation be to support private token sampling?* We find that the required accuracy de-

Method	Chebyshev Depth	Vocabulary Size		
		4k	8k	16k
Concurrent work (Rho et al., 2025b)	7	6.62 min (0.099 s/token)	13.20 min (0.099 s/token)	26.33 min (0.099 s/token)

Method	Chebyshev Depth	Vocabulary Size			
		16k	32k	64k	128k
ARGMAX (Zhang et al., 2024)	7	14.22 s (889 μs/token)	15.40 s (481 μs/token)	17.84 s (279 μs/token)	24.63 s (192 μs/token)
Hyperion (ours)	7	115 ms (7.2 μs/token)	142 ms (4.4 μs/token)	309 ms (4.8 μs/token)	605 ms (4.7 μs/token)
	8	222 ms (13.9 μs/token)	249 ms (7.8 μs/token)	537 ms (8.4 μs/token)	1128 ms (8.8 μs/token)
	9	386 ms (24.1 μs/token)	423 ms (13.2 μs/token)	1051 ms (16.4 μs/token)	2134 ms (16.7 μs/token)
	10	991 ms (61.9 μs/token)	1069 ms (33.4 μs/token)	2095 ms (32.7 μs/token)	4548 ms (35.5 μs/token)

Table 2: Token Sampling Latency (total time, time per-token in parentheses).

pends on the entropy (i.e. randomness) of the distribution. This entropy is typically summarized by a metric called *perplexity* and is controlled by the temperature parameter. Intuitively, perplexity measures how diffused the probability mass is: higher perplexity spreads probability across more tokens, while lower perplexity concentrates mass on a few dominant outcomes. As a result, higher perplexity distributions are more likely to cause the inverse transform sampling to evaluate the SIGN approximation on values near zero within its indistinguishable region $[-2^{-\epsilon}, 2^{-\epsilon}]$. The opposite holds for lower perplexity distributions.

Figure 3 quantifies this effect by showing the probability that Hyperion evaluates the SIGN approximation within the indistinguishable region for at least one value across the vocabulary. Several trends emerge from the results. First, increasing the depth of the Chebyshev approximation consistently reduces this probability. Second, as shown in Figure 3a, the vocabulary size has little effect on the likelihood of taking the SIGN approximation over the indistinguishable region. Finally, Figure 3b demonstrates that higher perplexity substantially increases this probability; for a Chebyshev depth of 8, it rises from approximately 4% at a perplexity of 2 to about 18% at a perplexity of 10. Although these probabilities may appear relatively high, evaluating the SIGN approximation within the indistinguishable region does not necessarily imply poor accuracy in Hyperion's output. As we show in our experimental evaluation (Section 5), the resulting L_1 error remains small in practice, indicating that Hyperion can tolerate such events without materially degrading sampling accuracy.

5 Experimental Evaluation

We implemented Hyperion on FIDESlib v1.0.0 (Agulló-Domingo et al., 2025), a GPU-accelerated implementation of OpenFHE (Al Badawi et al., 2022), with ring dimension $N = 2^{16}$ at 128-bit security and parameters provisioned to support the required multiplicative depth. For a fair latency comparison, we compared with a baseline ARGMAX (Zhang et al., 2024) and a concurrent work (Rho et al., 2025b) using the same SIGN approximation and NVIDIA L40s Tensor Core GPU. We evaluate both the efficiency and accuracy of Hyperion across a range of realistic vocabulary sizes, perplexities, and SIGN approximation depths. Typical vocabulary sizes in modern LLMs range from 32,000 (Llama) and 50,257 (GPT-3) up to around 100,000 (GPT-4). Typical LLM perplexities fall in a relatively small range, often between 2 and 10 (Ren et al., 2024; Wu et al., 2025), which we controlled by tuning the temperature parameter.

Table 2 compares the latency of concurrent work by Rho *et al.* (Rho et al., 2025b), ARGMAX (Zhang et al., 2024), and Hyperion across varying vocabulary sizes and Chebyshev multiplicative depths. Higher Chebyshev multiplicative depth increases Hyperion's runtime. Across all evaluated settings, Hyperion achieves substantially lower latency than both baseline approaches.

For a Llama-scale vocabulary of 32,000 tokens, Hyperion completes sampling in 0.14 seconds

Step	No CombineCtxts			With CombineCtxts		
	32k tokens	64k tokens	128k tokens	32k tokens	64k tokens	128k tokens
Precompute Plaintext (on CPU)	0.1815	0.1810	0.1822	0.2761	0.3978	0.4002
CDF (MultiRotateAndSum)	0.0315	0.0633	0.1545	0.0312	0.0671	0.1420
CombineCtxts	–	–	–	0.0037	0.0055	0.0086
Subtract Random	0.0003	0.0005	0.0014	0.0002	0.0005	0.0007
Sign	0.4456	0.8967	2.1672	0.2139	0.4618	0.9723
Rotate and Subtract	0.0022	0.0041	0.0095	0.0006	0.0022	0.0047
Precomputation + Eval. Latency	0.6609	1.146	2.515	0.5256	0.9349	1.5285
Eval. Latency (Hyperion)	0.4795	0.9647	2.3326	**0.2495**	**0.5370**	**1.1282**
Precompute Plaintext Speedup	1.38×	1.18×	1.08×	**2.14×**	**1.77×**	**1.35×**
CombineCtxts Speedup	–	–	–	**1.92×**	**1.79×**	**2.07×**

Table 3: Latency Breakdown for Hyperion with Chebyshev depth of 8 in seconds. Evaluation latency excludes plaintext precomputation and includes only the remaining runtime components. Plaintext precomputation speedup is computed relative to a baseline that includes plaintext computation time, while CombineCtxts speedup is computed relative to a baseline that does not use CombineCtxts.

($4.4\,\mu$s per token), whereas the baseline Argmax approach requires 15.4 seconds when using the same Chebyshev multiplicative depth of 7, yielding an improvement of over $100\times$. Even when increasing the Chebyshev depth to 9 to achieve higher accuracy, Hyperion runs in only 0.423 seconds ($13.2\,\mu$s per token), remaining significantly faster than Argmax at the same vocabulary size.

We further compare against the concurrent work of Rho *et al.* (Rho et al., 2025b) by implementing their CDF using a standard matrix multiplication algorithm with Baby-Step Giant-Step and hoisting optimizations. However, this approach scales poorly with larger vocabularies due to its requirement of $O(n)$ plaintext multiplications and $O(\sqrt{n})$ rotations for n tokens. As a concrete example, computing the CDF for 16,000 tokens requires 16,000 plaintext multiplications and 251 rotations, resulting in an end-to-end latency of 26.33 minutes. In contrast, Hyperion requires no plaintext multiplications and only 14 rotations, achieving an end-to-end latency of 115 ms, which is over $10{,}000\times$ faster.

Table 3 presents a latency breakdown of Hyperion using a Chebyshev multiplicative depth of 8, highlighting the impact of plaintext precomputation and CombineCtxts. The dominant source of latency is the Sign approximation, with only a small fraction of time spent computing the CDF. Consequently, improving slot utilization via CombineCtxts halves the cost of the Sign approximation (e.g. 0.8967 s vs. 0.4618 s for 64,000 tokens) and yields an overall speedup of nearly $2\times$. Finally, precomputing plaintexts (which are performed on CPU) further reduces Hyperion's evaluation latency by up to $2.14\times$ (e.g. from 0.5256 s to 0.2495 s for 32,000 tokens with CombineCtxts).

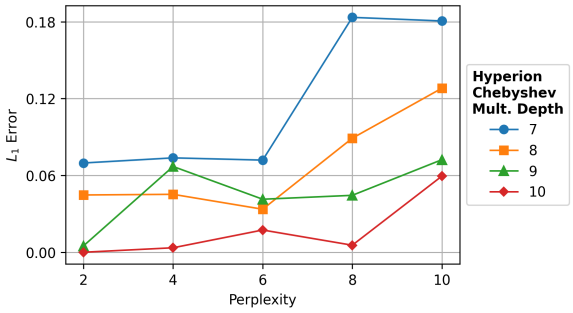


Figure 4: L_1 error vs. perplexity over deeper Chebyshev approximations with a vocabulary size of 32,000.

Figure 4 illustrates how perplexity impacts the accuracy of our method, measured by the L_1 distance from a one-hot encoding. For a vocabulary of 32,000 tokens, an L_1 distance of 0.04 corresponds to an average per-entry deviation of $\frac{0.04}{32000} = 1.25 \times 10^{-6}$ from the ideal one-hot vector. Consistent with Figure 3, Figure 4 shows that lower perplexity distributions result in smaller L_1 error under private token sampling. Despite the seemingly large probabilities suggested earlier, the empirical L_1 errors observed in Figure 4 are small in practice. Increasing the approximation depth further reduces error: at perplexity $= 2$, a depth of 10 achieves an L_1 error of 2.6×10^{-5}. At higher perplexities (e.g., 10), however, the error grows substantially and may be impractical, reaching $L_1 = 0.18$ and $L_1 = 0.12$ for depths 7 and 8, respectively. Whether moderate errors (e.g., $L_1 \approx 0.02$–0.05) are acceptable for end-to-end private LLM inference remains an open question for future work.

6 Conclusion

In this work, we presented Hyperion, an efficient, non-interactive algorithm for private token sampling. We plan to integrate this into a larger private LLM system and enable full private LLM token generation under homomorphic encryption. Even under conservative estimates, Hyperion can save around 11% of EncryptedLLM's (De Castro et al., 2025) forward pass latency. An open research question is what tolerance of error still enables private LLM inference.

Limitations

Hyperion explores private token sampling for LLM inference under homomorphic encryption (HE), but we do not integrate Hyperion into an existing end-to-end private LLM system (Moon et al., 2025; Zhang et al., 2024; Park et al., 2025; De Castro et al., 2025; Rovida and Leporati, 2024; Rho et al., 2025a; Zhang et al., 2025). This is largely due to architectural and practical mismatches with prior work. Several systems (Moon et al., 2025; Park et al., 2025; Rovida and Leporati, 2024; Rho et al., 2025a; Lim et al., 2025) target encoder-only transformer models, whereas text generation is typically performed using decoder-only architectures. Other works do not release open-source implementations (De Castro et al., 2025), or their released codebases do not provide a complete end-to-end system (Zhang et al., 2024). Additionally, some systems (Zhang et al., 2024, 2025) assume batched inference across many users or many inputs that share cryptographic keys whereas Hyperion does not.

As a result, an open question for future work is how much approximation error introduced by Hyperion can be tolerated while still enabling accurate end-to-end private LLM inference. Integrating Hyperion into a full system may also introduce additional challenges, such as cumulative CKKS approximation error. Moreover, our work does not quantify the total computational cost of a complete end-to-end private LLM pipeline.

Finally, Hyperion does not currently support private top-k or top-p (nucleus) sampling, as explored in concurrent work (Avitan et al., 2025), which may better reflect common LLM decoding strategies. Applying their techniques is nontrivial and requires deeper, more expensive computation than Hyperion. Nonetheless, we note that adjusting the temperature to reduce perplexity induces behavior similar to top-p sampling.

Acknowledgments

This work was supported in part by a Google PSS Privacy Research Faculty Award and the Air Force Research Laboratory, Information Directorate in Rome, NY. The authors would like to thank them for their generous support.

References

Carlos Agulló-Domingo, Óscar Vera-López, Seyda Guzelhan, Lohit Daksha, Aymane El Jerari, Kaustubh Shivdikar, Rashmi Agrawal, David Kaeli, Ajay Joshi, and José L. Abellán. 2025. FIDESlib: A Fully-Fledged Open-Source FHE Library for Efficient CKKS on GPUs. In *2025 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Ghent, Belgium. IEEE. Poster paper.

Ahmad Al Badawi, Jack Bates, Flavio Bergamaschi, David Bruce Cousins, Saroja Erabelli, Nicholas Genise, Shai Halevi, Hamish Hunt, Andrey Kim, Yongwoo Lee, Zeyu Liu, Daniele Micciancio, Ian Quah, Yuriy Polyakov, Saraswathy R.V., Kurt Rohloff, Jonathan Saylor, Dmitriy Suponitsky, Matthew Triplett, and 2 others. 2022. Openfhe: Open-source fully homomorphic encryption library. In *Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, WAHC'22, page 53–63, New York, NY, USA. Association for Computing Machinery.

Matan Avitan, Moran Baruch, Nir Drucker, Itamar Zimerman, and Yoav Goldberg. 2025. Efficient decoding methods for language models on encrypted data. In *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, pages 1777–1794, Mumbai, India. The Asian Federation of Natural Language Processing and The Association for Computational Linguistics.

Jean-Philippe Bossuat, Christian Mouchet, Juan Troncoso-Pastoriza, and Jean-Pierre Hubaux. 2021. Efficient bootstrapping for approximate homomorphic encryption with non-sparse keys. In *Advances in Cryptology – EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part I*, page 587–617, Berlin, Heidelberg. Springer-Verlag.

E. W. Cheney. 1966. *Introduction to approximation theory*. International series in pure and applied mathematics. McGraw-Hill Book Co., New York.

Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic encryption for arithmetic of approximate numbers. In *Advances in Cryptology – ASIACRYPT 2017*, pages 409–437, Cham. Springer International Publishing.

Jung Hee Cheon, Dongwoo Kim, and Duhyeong Kim. 2020. Efficient homomorphic comparison methods with optimal complexity. In *Advances in Cryptology – ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part II*, page 221–256, Berlin, Heidelberg. Springer-Verlag.

Jung Hee Cheon, Dongwoo Kim, Duhyeong Kim, Hun Hee Lee, and Keewoo Lee. 2019. Numerical method for comparison on homomorphically encrypted numbers. In *Advances in Cryptology – ASIACRYPT 2019: 25th International Conference on the Theory and Application of Cryptology and Information Security, Kobe, Japan, December 8–12, 2019, Proceedings, Part II*, page 415–445, Berlin, Heidelberg. Springer-Verlag.

Wonhee Cho, Guillaume Hanrot, Taeseong Kim, Minje Park, and Damien Stehlé. 2024. Fast and accurate homomorphic softmax evaluation. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, CCS '24, page 4391–4404, New York, NY, USA. Association for Computing Machinery.

Leo De Castro, Daniel Escudero, Adya Agrawal, Antigoni Polychroniadou, and Manuela Veloso. 2025. Encryptedllm: privacy-preserving large language model inference via gpu-accelerated fully homomorphic encryption. In *Proceedings of the 42nd International Conference on Machine Learning*, ICML'25. JMLR.org.

Seunghu Kim, Eymen Ünay, Ayse Yilmazer-Metin, and Hyung Tae Lee. 2025. Optimized rank sort for encrypted real numbers. Cryptology ePrint Archive, Paper 2025/1170.

Eunsang Lee, Joon-Woo Lee, Jong-Seon No, and Young-Sik Kim. 2022. Minimax approximation of sign function by composite polynomial for homomorphic comparison. *IEEE Transactions on Dependable and Secure Computing*, 19(6):3711–3727.

Joon-Woo Lee, Eunsang Lee, Yongwoo Lee, Young-Sik Kim, and Jong-Seon No. 2020. Optimal minimax polynomial approximation of modular reduction for bootstrapping of approximate homomorphic encryption. *IACR Cryptol. ePrint Arch.*, 2020:552.

Lawrence Lim, Vikas Kalagi, Divyakant Agrawal, and Amr El Abbadi. 2025. Tricycle: Private transformer inference with tricyclic encodings. Cryptology ePrint Archive, Paper 2025/1200.

Federico Mazzone, Maarten Everts, Florian Hahn, and Andreas Peter. 2025. Efficient ranking, order statistics, and sorting under {CKKS}. In *34th USENIX Security Symposium (USENIX Security 25)*, pages 8541–8558.

Jungho Moon, Dongwoo Yoo, Xiaoqian Jiang, and Miran Kim. 2025. Thor: Secure transformer inference with homomorphic encryption. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security*, CCS '25, page 3765–3779, New York, NY, USA. Association for Computing Machinery.

Dongjin Park, Eunsang Lee, and Joon-Woo Lee. 2025. 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)*, pages 11090–11111, Vienna, Austria. Association for Computational Linguistics.

Xuan Ren, Biao Wu, and Lingqiao Liu. 2024. I learn better if you speak my language: Understanding the superior performance of fine-tuning large language models with LLM-generated responses. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 10225–10245, Miami, Florida, USA. Association for Computational Linguistics.

Donghwan Rho, Taeseong Kim, Minje Park, Jung Woo Kim, Hyunsik Chae, Ernest K. Ryu, and Jung Hee Cheon. 2025a. Encryption-friendly LLM architecture. In *The Thirteenth International Conference on Learning Representations*.

Donghwan Rho, Sieun Seo, Hyewon Sung, Chohong Min, and Ernest K. Ryu. 2025b. Traveling salesman-based token ordering improves stability in homomorphically encrypted language models. *Preprint*, arXiv:2510.12343.

Lorenzo Rovida and Alberto Leporati. 2024. Transformer-based language models and homomorphic encryption: An intersection with bert-tiny. In *Proceedings of the 10th ACM International Workshop on Security and Privacy Analytics*, IWSPA '24, page 3–13, New York, NY, USA. Association for Computing Machinery.

Chao-Chung Wu, Zhi Rui Tam, Chieh-Yen Lin, Yun-Nung Chen, Shao-Hua Sun, and Hung yi Lee. 2025. Mitigating forgetting in LLM fine-tuning via low-perplexity token learning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*.

Yi-Leh Wu, Divyakant Agrawal, and Amr El Abbadi. 2001. Applying the golden rule of sampling for query estimation. *SIGMOD Rec.*, 30(2):449–460.

Jiawen Zhang, Xinpeng Yang, Lipeng He, Kejia Chen, Wen jie Lu, Yinghao Wang, Xiaoyang Hou, Jian Liu, Kui Ren, and Xiaohu Yang. 2024. Secure transformer inference made non-interactive. Cryptology ePrint Archive, Paper 2024/136.

Linru Zhang, Xiangning Wang, Jun Jie Sim, Zhicong Huang, Jiahao Zhong, Huaxiong Wang, Pu Duan, and Kwok Yan Lam. 2025. MOAI: Module-optimizing architecture for non-interactive secure transformer inference. Cryptology ePrint Archive, Paper 2025/991.