Efficient Decoding Methods for Language Models on Encrypted Data

Matan Avitan[1,2] **Moran Baruch**[1] **Nir Drucker**[1]
Itamar Zimerman[1,3] **Yoav Goldberg**[2,4]

[1]IBM Research [2]Bar-Ilan University
[3]Tel Aviv University [4]Allen Institute for Artificial Intelligence

Abstract

Large language models (LLMs) power modern AI applications, but processing sensitive data on untrusted servers raises privacy concerns. Homomorphic encryption (HE) enables computation on encrypted data for secure inference. However, neural text generation requires decoding methods like argmax and sampling, which are non-polynomial and thus computationally expensive under encryption, creating a significant performance bottleneck. We introduce CutMax, an HE-friendly argmax algorithm that reduces ciphertext operations compared to prior methods, enabling practical greedy decoding under encryption. We also propose the first HE-compatible nucleus (top-p) sampling method, leveraging CutMax for efficient stochastic decoding with provable privacy guarantees. Both techniques are polynomial, supporting efficient inference in privacy-preserving settings. Moreover, their differentiability facilitates gradient-based sequence-level optimization as a polynomial alternative to straight-through estimators. We further provide strong theoretical guarantees for CutMax, proving its convergence via exponential amplification of the gap ratio between the maximum and runner-up elements. Evaluations on realistic LLM outputs show latency reductions of $24\times$–$35\times$ over baselines, advancing secure text generation.

1 Introduction

Recent advances in LLMs have enabled the development of powerful AI systems capable of generating fluent text at scale (Brown et al., 2020). However, deploying these models in real-world applications often involves sending sensitive user data, such as personal messages or medical records, to remote servers, raising significant privacy concerns (Yao et al., 2024; Yan et al., 2024). Despite major progress in efficient encrypted inference, extending these methods to generative LLMs remains an open challenge due to the non-polynomial nature of decoding operations. Homomorphic encryption (HE) offers a promising solution by allowing computations on encrypted data, ensuring that servers can process queries without accessing their plaintext content (Gentry, 2009). Under HE, users encrypt their inputs, the server performs computations (e.g., running an LLM), and returns an encrypted result that only the user can decrypt.

In this paper, we present differentiable and polynomial argmax and nucleus sampling algorithms tailored for encrypted LLM decoding. Unlike prior approaches, such as that of Bengio et al. (2013), which use argmax in the forward pass and estimate gradients in the backward pass via straight-through estimators (STE), our methods are fully differentiable in their plaintext form (without approximations) and remain so under polynomial HE approximations. This differentiability, together with their polynomial form, makes them well suited to AI-privacy settings such as HE; the plaintext formulations are exactly differentiable, and the HE instantiations preserve differentiability via polynomial approximations (see Sect. 3.2). Most HE schemes, including the CKKS scheme used in this work (Cheon et al., 2017), support only polynomial operations like addition and multiplication. This limitation necessitates the design of *HE-friendly* algorithms that rely exclusively on such operations, posing a challenge for text generation tasks that involve complex decoding steps. While secure inference on LLMs using HE has been demonstrated (Zimerman et al., 2024a; de Castro et al., 2024; Zhang et al., 2024a), extending these techniques to multi-token text generation remains challenging. Decoding methods like argmax (for greedy decoding) and sampling (e.g., nucleus sampling (Holtzman et al., 2019a)) are computationally inefficient or impractical in the encrypted domain due to their reliance on non-polynomial operations.

State-of-the-art homomorphic argmax implementations follow two comparison-heavy patterns:

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

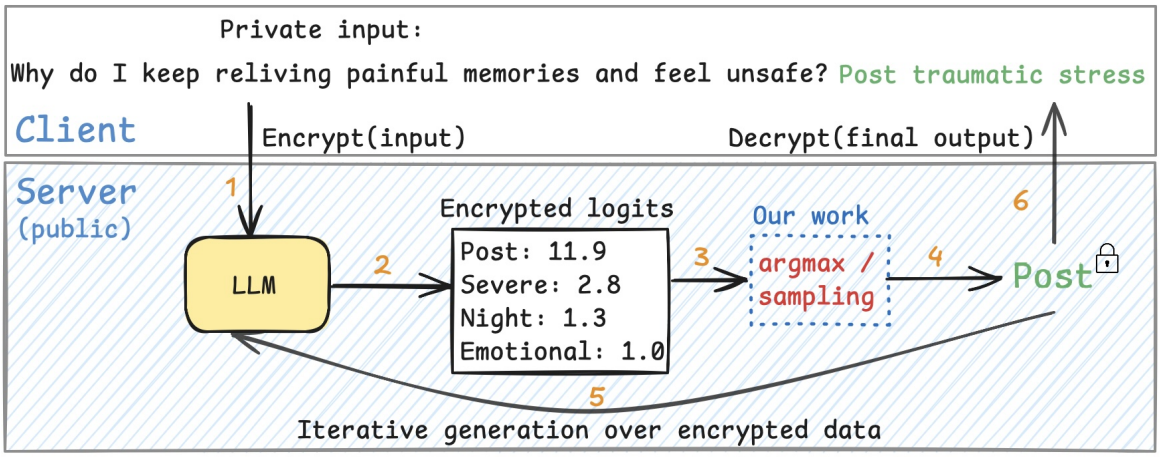


Figure 1: **Scope**: Secure LLM generation over the clients' encrypted data using HE (model is not necessarily encrypted). Here, standard decoding methods like argmax and sampling (**red**) for selecting the next token are not HE-friendly or considered inefficient. We introduce efficient and scalable HE-friendly methods to evaluate them under encryption.

a *tournament* tree and an all-to-all *league* schedule, both realized via deep polynomial approximations of SIGN (Chakraborty and Zuber, 2022; Crawford et al., 2018; Folkerts and Tsoutsos, 2024; Jovanovic et al., 2022). A detailed illustration of these designs is provided in Figure 6 (Sect. C), which highlights their sequential comparison stages and reliance on costly SIGN evaluations.

CutMax removes comparisons altogether. Each of its T iterations applies two global reductions (mean and variance), an inverse-square-root (InvSqr), an elementwise odd power, and a final normalization (Sect. 3, Algorithm 1, Sect. 3.2, Algorithm 2). Consequently, the number of *sequential* stages is a small constant T—*weakly dependent* on the vector length n (e.g., the vocabulary size)—whereas tournament and league require $\log_2 n$ and n sequential rounds, respectively, each performing encrypted comparisons via polynomial approximations of SIGN. Because InvSqr and division (Inv) are shallower and faster than SIGN at similar accuracy (Table 3), CutMax achieves much lower depth and fewer bootstraps per token. Empirically, T is a small constant (e.g., $T \leq 3$–4), consistent with the latency gains we observe for vocabularies of $|\mathcal{V}| \approx 3.3 \times 10^4$ and 1.5×10^5 (Sect. 4, Table 1). Implementation-level costs such as rotations under SIMD packing are deferred to Sect. 3.2 and Sect. D.

Our contributions address key bottlenecks in secure text generation, as illustrated in Figure 1, and include: (i) **CutMax**. An efficient, homomorphic encryption (HE)-friendly argmax algorithm that enables practical greedy decoding under encryption with fewer ciphertext operations than previous methods. (ii) **Encrypted nucleus sampling**. The first HE-compatible nucleus (top-p) sampling method, enables stochastic decoding with provable privacy guarantees, via CutMax. (iii) **Theoretical convergence guarantees**. A formal proof establishing that CutMax converges via exponential amplification of the gap ratio between the maximum and runner-up elements, providing a rigorous foundation for its rapid convergence in a small number of iterations, weakly dependent on the array size. (iv) **Differentiable decoding primitives**. Implementations of argmax and nucleus sampling that are real-analytic and fully differentiable in plaintext, leveraging smooth operations like $1/\sqrt{x}$ (and polynomial-only in their HE approximations) to enable exact gradient-based sequence-level training as a theoretically grounded alternative to straight-through estimators. By enabling efficient and accurate multi-token generation under full encryption, our work advances the deployment of privacy-preserving LLMs in real-world settings, bridging a critical gap in secure AI systems.

2 Background: Homomorphic Argmax

State-of-the-art homomorphic argmax implementations for LLM decoding rely on comparison-heavy designs, primarily the *tournament* tree and *league* schedule, both implemented via deep polynomial approximations of the SIGN function (Chakraborty and Zuber, 2022; Crawford et al., 2018; Folkerts and Tsoutsos, 2024; Jovanovic et al.,

2022; Zhang et al., 2024b, 2025a). In the tournament design, approximately $n-1$ pairwise comparisons are organized into $\log_2 n$ sequential stages, halving the number of candidates per stage until the maximum remains. The league design performs $\binom{n}{2} = \Theta(n^2)$ comparisons across n sequential rounds, accumulating scores such that the maximum achieves a score of $n-1$. Both methods are illustrated in Figure 6 (Sect. C). Despite leveraging CKKS SIMD parallelism to execute multiple comparisons within a round, these approaches incur high multiplicative depth due to repeated SIGN evaluations, leading to costly bootstrapping and limiting scalability for large vocabularies ($n \sim 10^5$). Some works, such as Chakraborty and Zuber (2022); Folkerts and Tsoutsos (2024); Grivet Sébert et al. (2021), use TFHE (Chillotti et al., 2016), restricting input sizes (e.g., $n \leq 256$) due to efficiency constraints. Hybrid approaches combine tournament and league methods to exploit SIMD packing (Iliashenko and Zucca, 2021; Mazzone et al., 2025), but still rely on slow SIGN approximations. As shown in Table 3 (Sect. B), SIGN evaluations are significantly deeper and slower than inverse or inverse-square-root operations, which our CutMax algorithm uses instead (Sect. 3.2). These limitations make prior methods impractical for efficient, privacy-preserving LLM decoding, motivating our polynomial-based approach that eliminates comparisons entirely.

2.1 HE Preliminaries

HE enables computation on encrypted data without requiring decryption (Gentry, 2009). Informally, it is defined as follows: Let $\mathcal{R}_1(+,\cdot), \mathcal{R}_2(\oplus,\odot)$ be two rings for the plaintext and ciphertext spaces, respectively. Given plaintext inputs $m_1, m_2 \in \mathcal{R}_1$, their encryption $[\![m_i]\!] := \text{Enc}(m_i) \in \mathcal{R}_2$, satisfies the following correctness and homomorphic properties:

- $\text{Dec}([\![m_1]\!]) = m_1 + \epsilon$
- $\text{Dec}([\![m_1]\!] \oplus [\![m_2]\!]) = m_1 + m_2 + \epsilon$
- $\text{Dec}([\![m_1]\!] \odot [\![m_2]\!]) = m_1 \cdot m_2 + \epsilon$

where ϵ is a small noise introduced by the HE scheme, similar to the noise accumulated in floating-point computations. HE is *semantically secure* and thus every call to $Enc(m_1)$ results with a new pseudo random ciphertext. Details about the standard *threat-model* of HE-based applications are provided in Sect. A. As HE only supports *polynomial operations* (additions and multiplications), standard transformer components like Softmax and LayerNormalization are not directly supported and must be approximated or replaced. Prior-art such as (Gilad-Bachrach et al., 2016; Aharoni et al., 2023; Baruch et al., 2022, 2024; Zimerman et al., 2024b,a; de Castro et al., 2024; Zhang et al., 2024a) have proposed polynomial-friendly adaptations for encrypted inference. We focus on the *generation process* where new challenges arise: decoding operations such as argmax and sampling are not polynomial.

Algorithm 1 CutMax

Require: $X = (X_1, \ldots, X_n)$, and $p, T \in \mathbb{Z}_+$, where p is odd and $c \in \mathbb{R}_+$.
Ensure: Z - one-hot approx. of $\text{argmax}(X)$.

$Y^{(1)} = X$
for $t = 1$ **to** T **do**
  $\mu = \frac{1}{n}\sum_{i=1}^{n} Y_i^{(t)}$ ▷ Mean of $Y^{(t)}$
  $\sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(Y_i^{(t)} - \mu)^2$ ▷ Variance of $Y^{(t)}$
  $Y^{(t+1)} = \left(\frac{Y_i^{(t)} - \mu}{c \cdot \sqrt{\sigma^2}} + 1\right)_{i\in[n]}$ ▷ Standardization
  $Y^{(t+1)} = \left(Y^{(t+1)}\right)^p_{i\in[n]}$
$Z = \left(\frac{Y_i^{(T)}}{\sum_j Y_j^{(T)}}\right)_{i\in[n]}$
return Z

3 Method

This section presents our polynomial algorithms for argmax and nucleus sampling, tailored for HE.

3.1 The CutMax Algorithm

Intuitively, CutMax repeatedly 'stretches' the distribution of values and cuts off the lower part, such that after a few iterations only the highest value remains significantly non-zero (see Figure 3). CutMax is inspired by standardization in statistics (subtracting the mean and dividing by standard deviation), alongside the idea of centering the array around 1; therefore, when taking an odd power of the standardized values, values smaller than the mean would vanish while values greater than the mean would amplify. The algorithm's fast convergence is due to the right skewness induced by the power operation.

Algorithm 1 provides the pseudo-code for CutMax, which takes as input a vector X of n elements, an odd integer p, a constant scaling factor $c > 0$, and the maximum number of iterations T. For instance, one instantiation uses $p = 11$,

$c = 5$, and $T = 3$. We denote the index set as $[n] = \{1, \ldots, n\}$. CutMax is an iterative algorithm that runs for at most T iterations or until convergence. Each iteration t consists of two steps: **standardization** and **distance amplification**.

Standardization (Lines 3-5). The normalization process starts by computing the mean μ of the current vector state $Y^{(t)}$, the variance σ^2, then the inverse standard deviation $\frac{1}{\sqrt{\sigma^2}}$. Subsequently, it scales $Y^{(t)}$ by subtracting the mean and multiplying by $\frac{1}{c \cdot \sigma}$. The choice of c controls how much we "flatten" the distribution before raising the power. A larger c means we divide by a larger number, making the normalized values smaller in magnitude.

Distance Amplification (Line 6). We raise each normalized element to an odd power p, which preserves the sign of $Y_i - \mu$. Elements above (resp. below) the mean ($Y_i - \mu > 0$) remain positive (resp. negative). The effect of this power raising is twofold: it shrinks the magnitude of small values and amplifies the larger values, where

$$\left|Y_i^{(t)}\right| < 1 \Longrightarrow \left|Y_i^{(t)}\right| \gg \left|Y_i^{(t)}\right|^p \tag{1}$$

$$\left|Y_i^{(t)}\right| > 1 \Longrightarrow \left|Y_i^{(t)}\right| \ll \left|Y_i^{(t)}\right|^p \tag{2}$$

By appropriately scaling $Y^{(t)}$ before this step, we ensure that all but the largest elements satisfy $|Y_i| < 1$, causing them to shrink, while the largest (and possibly a few near-largest) elements may have $|Y_i| > 1$ and thus grow. This drastically amplifies the gap between the maximum and the rest.

Convergence. As formally proven in Appendix E, each CutMax iteration unconditionally grows the gap between the maximum and runner-up elements, yielding exponential amplification of their ratio and ensuring convergence under a mild δ-gap condition on the input. Beyond this theoretical contraction, CutMax also prunes much more aggressively than comparison-based schemes. Unlike the tournament argmax, which discards only half of the $Y^{(t)}$ values per stage, CutMax zeroes out a *large majority* each round: empirically, after iteration t, the empirical CDF at the mean satisfies $\mathrm{CDF}_{Y^{(t)}}(\mu^{(t)}) \gg \frac{1}{2}$, so far more than half of the $Y_i^{(t)}$ fall below the mean and are driven toward zero by the subsequent odd-power step (Figure 2).

As established by Theorem E.10, these properties guarantee global convergence to the fixed point

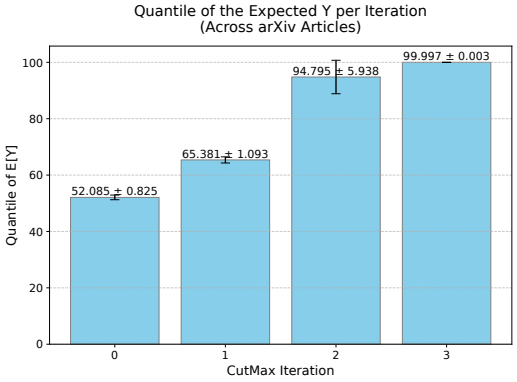


Figure 2: **Evolution of Logits Below the Mean Across CutMax Iterations** (After Algorithm 1, Algorithm 1). We ran CutMax for $T = 3$ with the best hyperparameters $(p, c) = (19, 27)$, found via grid search, on 1,000 arXiv articles passed through GPT-2 (vocabulary size $|\mathcal{V}| = 50{,}257$ (Radford et al., 2019)). Bars show the mean±std of the fraction of entries $Y_i^{(t)} < \mathbb{E}[Y^{(t)}]$ at each iteration t.

under mild conditions; for the practical hyperparameters we use, the limit is nearly one-hot. This explains why only a small, nearly constant number of iterations (e.g., $T \leq 3$–4) suffices in practice, regardless of the initial logit distribution (Sect. 4).

3.2 HE-Friendly CutMax

Building on the HE preliminaries in Sect. 2, we adapt CutMax to be fully polynomial by approximating $1/\sigma$ (Line 5, Algorithm 1) and $1/\sum_j Y_j^{(T)}$ (Line 7) using Goldschmidt methods (Goldschmidt, 1964) (details in Sect. B.1). This yields Algorithm 2, the FHE variant. Notably, having a polynomial algorithm is not enough, we also need to ensure that no overflows or precision issues compromise the correctness of the results. Specifically, we must prove the following: **(i) No overflows.** Let B_{CKKS} be the bound configured by the scheme on the unencrypted inputs, beyond which an overflow may occur. Then, all intermediate values produced by the algorithm must remain below B_{CKKS}. **(ii) No approximation errors.** The InvSqr and Inv approximations assume inputs lie within a certain range, e.g., $[\frac{1}{2^n}, 1]$ for some small n. If an input x falls in a wider range $[a, b]$, we instead compute $\frac{\text{InvSqr}(\frac{x}{b})}{b}$. We must show that $\frac{a}{b} > \frac{1}{2^n}$; otherwise, the approximation may yield incorrect results. Sect. 2 reviewed the tournament method that requires $O(n)$ comparisons in $\log n$ sequential stages and the league method that requires n^2 comparisons in n sequential stages. In contrast, CutMax

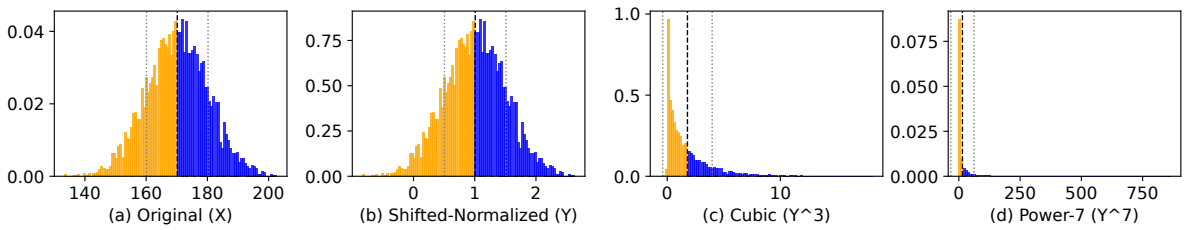


Figure 3: **Illustration of CutMax First Iteration on a Normal Distribution** CutMax algorithm first iteration illustration, applied to the original random variable $X \sim \text{Normal}(170, 10)$ (a). We first standardize and shift X using $Y = \frac{X-\mu_X}{c\sigma_X} + 1$ with $c = 5$ (b), then raise Y to the power $p = 3$ (c) or $p = 7$ (d) to induce right skew. The resulting distribution $Z = Y^7$ (d) has a maximum normalized height $\max\left(Z/\sum_i Z_i\right) = 0.0033$, indicating a strong right-tail skew. A vertical black line indicates the mean of every distribution and gray lines indicate the first standard deviation.

Algorithm 2 CutMaxHE - Encrypted CutMax

Require: $[\![X]\!] = [\![(X_1, \ldots, X_n)]\!]$, and $p, T \in \mathbb{Z}_+$, where p is odd, and $c \in \mathbb{R}_+$.
Ensure: $[\![Z]\!]$ one-hot approx. of $\text{argmax}(X)$.

1: $[\![Y^{(1)}]\!] = [\![X]\!]$
2: **for** $t = 1$ **to** T **do**
3: $\quad [\![\mu]\!] = \frac{1}{n}\,\text{RotAndSum}([\![Y^{(t)}]\!])$
4: $\quad [\![\sigma^2]\!] = \frac{1}{n}\,\text{RotAndSum}\left(([\![Y^{(t)}]\!] \ominus [\![\mu]\!])^2\right)$
5: $\quad [\![\sigma^{-1}]\!] = \text{InvSqr}([\![\sigma^2]\!])$
6: $\quad [\![Y^{(t+1)}]\!] = \left(\frac{1}{c}[\![\sigma^{-1}]\!] \odot ([\![Y^{(t)}]\!] \ominus [\![\mu]\!])\right) \oplus 1$
7: $\quad [\![Y^{(t+1)}]\!] = \left([\![Y^{(t+1)}]\!]\right)^p$
8: $[\![Z]\!] = [\![Y^{(T)}]\!] \odot \text{Inv}\left(\text{RotAndSum}([\![Y^{(T)}]\!])\right)$
9: **return** $[\![Z]\!]$

requires only T InvSqr and 1 Inv operations. Table 3 in Sect. B (taken from (Moon et al., 2024a)) shows that the InvSqr implementation is 1.5–2.1× faster than the SIGN implementation used in all prior art, at the same error level. In addition, T is weakly-dependent on n and much smaller. Consequently, CutMax outperforms prior-art. Unfortunately, the situation is not that simple. Modern HE schemes support single instruction multiple data (SIMD) operations, meaning a ciphertext can encrypt a vector of s elements instead of just a single element. In this case, multiplications ($\odot$), additions ($\oplus$), and subtractions ($\ominus$) are applied element-wise homomorphically over the vector. Furthermore, a homomorphic rotation by $\ell < s$ positions is also available (Cheon et al., 2017). We defer the discussion of how SIMD impacts latency to Sect. D, and here we complete the picture with an implementation of CutMax over HE. Algorithm 2 is the HE approximation variant of CutMax, where the input vector X is encrypted. For brevity, we assume $n = a \cdot s$, $a \in \mathbb{Z}_+$, and write the algorithm so that modern compilers such as HELayers (Aharoni et al., 2023) can automatically implement it. Lines 3, 4, and 8 of Algorithm 1 are efficiently computed in Algorithm 2 using the RotAndSum method (see (Adir et al., 2024)), which takes an encrypted vector $v = (v_1, \ldots, v_s)$ and returns a vector of s elements, each containing the sum $\sum v_i$. This requires $\log s$ rotations and additions. Multiplying by a plaintext scalar ($\frac{1}{n}$ at lines 3,4 and $\frac{1}{c}$ at line 6) is considered a cheap HE operation as well as the homomorphic subtractions at lines 4 and 6. Raising to the power of 2 (Line 4) and p (Line 6) require 1 and $\log p$ multiplications, respectively. Finally, the heaviest operations used are the InvSqr (Line 5) and Inv (Line 8) that use the Goldschmidt approximations (Sect. B.1). Denote by M_{Inv} (resp. D_{Inv}) and M_{InvSqr} (resp. D_{InvSqr}) the number of multiplications (resp. multiplication depth) required by the Inv and InvSqr approximations over one ciphertext, respectively. Then CutMaxHE executes

$$T \cdot (M_{\text{InvSqr}} + 5 + \log p) + M_{\text{Inv}} + 1 \qquad (3)$$

multiplications with a total depth of

$$T \cdot (D_{\text{InvSqr}} + 5 + \log p) + D_{\text{Inv}} + 1 \qquad (4)$$

and $O(T \log s)$ rotations.

3.3 HE-Friendly Nucleus Sampling

In text generation, language models produce a probability distribution over a large vocabulary at each time step. The next token is selected from this distribution using a decoding strategy. The simplest strategy is greedy decoding, where the token with the highest probability (argmax) is chosen.

More commonly, however, high-quality text generation relies on stochastic decoding methods that introduce controlled randomness to improve fluency, diversity, and coherence. Two widely used stochastic decoding methods are top-k sampling (Fan et al., 2018) and nucleus (top-p) sampling (Holtzman et al., 2019b). In top-k sampling, only the k most probable tokens are considered, and one is sampled according to its probability. In nucleus sampling, the model selects the smallest set of top-ranked tokens whose cumulative probability exceeds a threshold p, and samples from this adaptive set. These techniques prevent degeneration and repetition, making them essential in modern LLM-based systems.

Sampling under HE. As established, our CutMax algorithm is polynomial and thus compatible with HE; we confirm its correctness empirically in Sect. 4. Whereas CutMax handles deterministic decoding, we next extend it to stochastic sampling. The first naive attempt is to use the *inverse transform sampling* method, which generates random samples from a given probability distribution using its cumulative distribution function (CDF): Given an encrypted probability density function (PDF) X with CDF F_X, the method samples a plaintext value $U \sim \mathrm{Uniform}(0,1)$, and returns $x = F_X^{-1}(u)$, where F_X^{-1} is the inverse CDF (iCDF), and x is a sample from X. This sampling method is inefficient as it requires at least one HE heavy comparison (Sect. B) to sample an element.

Using Gumbel Distribution. We can improve upon the above by using the Gumbel distribution. We begin by recalling the relationship between the Gumbel and Uniform distributions, as well as the *Gumbel-max trick*.

Definition 3.1. *(Gumbel, 1954) Let* $U \sim \mathrm{Uniform}(0,1)$*, then* $G = -\log(-\log U)$ *has a Gumbel distribution* $G \sim \mathrm{Gumbel}(\mu = 0, \beta = 1)$.

Lemma 3.2 (Gumbel-Max Trick (Maddison et al., 2014))**.** *For* $X \in R^k$*, let* $G_i \sim \mathrm{Gumbel}(0,1)$ *be independent for each* i*. Then the random variable* $Y = \mathrm{argmax}_i\,(X_i + G_i)$ *follows a categorical distribution with probabilities given by,*

$$\mathbb{P}(Y = i) = \frac{e^{X_i}}{\sum_j e^{X_j}} = \mathrm{Softmax}(X_i)\,.$$

Using the above, our sampling method is as follows: for a given encrypted logits array $X \in \mathbb{R}^n$, sample unencrypted $U \in \mathrm{Uniform}(0,1)^n$ and convert it to $G \sim \mathrm{Gumbel}(0,1)^n$, then compute $\mathrm{CutMax}(X + G)$ over HE, which only requires fast additions and one call to CutMax.

Algorithm 3 Nucleus One-Shot Sampling

Require: Logits $[\![X]\!]$, $X \in \mathbb{R}^n$; Nucleus mass $p \in (0,1)$.
Ensure: Encrypted one-hot sample $[\![z]\!]$.
1: $\alpha = \frac{\ln(1-p)}{\ln(p)}$
2: $\beta = 1$
3: $U \leftarrow \mathrm{Uniform}(0,1)^n$
4: $G = \left(F^{-1}_{\mathrm{Beta}(\alpha,\beta)}(U_i)\right)_{i \le 1,\ldots,n}$ ▷ Unencrypted noise
5: $[\![\tilde{X}]\!] = [\![X]\!] \oplus G$ ▷ Perturb logits
6: **return** $\mathrm{CutMax}([\![\tilde{X}]\!])$

Efficient Nucleus Sampling. While our Gumbel-based method allows efficient sampling over HE, we can do better. We aim to develop an algorithm inspired by (Maddison et al., 2014) that will enable us to sample only from the top of the distribution, as in nucleus sampling, so that words from the tail of the logits distribution could not be sampled, and potentially disrupt the coherence of the generated text. To this end, we aim to sample G from a PDF that satisfy the condition $\mathbb{P}(G > p) = p$. Luckily, the $\mathrm{Beta}(\alpha, \beta)$ distribution has this property when $\beta = 1$.

Lemma 3.3. *Let* $G \sim \mathrm{Beta}(\alpha, 1)$*, then* $\mathbb{P}(G > p) = p$ *for* $\alpha = \frac{\ln(1-p)}{\ln(p)}$.

Proof. The CDF of $\mathrm{Beta}(\alpha,1)$ is $F_{\mathrm{Beta}(\alpha,1)}(x) = x^\alpha$. The requirement $\mathbb{P}(G > p) = p$ holds if and only if $F_{\mathrm{Beta}(\alpha,1)}(p) = 1 - p$. Thus, $p^\alpha = 1 - p$, or alternatively, $\alpha = \frac{\ln(1-p)}{\ln(p)}$. □

Observation 3.4. *Theorem 3.3 targets* $\mathbb{P}(G > p) = p$*. We can generalize this result by asking* $\mathbb{P}(G > p) = q$ *for any* $q \in [0,1]$*, which yields* $\alpha = \frac{\ln(1-q)}{\ln(p)}$

Algorithm 3 introduces a single-shot, nucleus-restricted sampling routine based on Theorem 3.3. It starts by draw $G \sim \mathrm{Beta}(\alpha, \beta)$ (Line 4). Subsequently, we form the "cut-noise" logits at Line 5, so that only those tokens whose cumulative mass exceeds p can potentially become the maximum. Finally, the algorithm executes a single invocation of CutMax at Line 6 to homomorphically recover the one-hot encrypted sample from the nucleus set. To summarize, we introduced a one-shot nucleus sampling algorithm whose polynomial formulation requires only a single CutMax evaluation under

HE, while maintaining exact distributional behavior and full HE compatibility.

3.4 Differentiability for Sequence Optimization

A key property of CutMax and our nucleus sampling is their composition from polynomial operations (additions, multiplications, means/variances, InvSqr, and elementwise powers). In the plaintext domain (Algorithm 1), these operations are naturally differentiable without any approximations: for example, the inverse-square-root $1/\sqrt{x}$ is C^∞ smooth and differentiable for $x > 0$, enabling exact gradient computation via automatic differentiation. This contrasts with prior tournament/league methods, which inherently rely on the discontinuous SIGN function, even in plaintext, SIGN requires smoothing or approximation for differentiability, often leading to high-degree polynomials that can introduce gradient instabilities.

In HE, both approaches use polynomial approximations, but CutMax approximates smooth functions like $1/\sqrt{x}$, which can be achieved with lower-degree polynomials compared to approximating the sharp discontinuity of SIGN (as evidenced by the lower multiplicative depths in Table 3). Theoretically, polynomial composition preserves differentiability, providing a rigorous foundation for exact end-to-end gradients through the full forward pass.

This stands in contrast to non-differentiable ops like discrete argmax or sampling, which require approximations like straight-through estimators (STE; Bengio et al. (2013)) during backpropagation—where gradients are passed 'straight through' the hard argmax as if it were a soft identity. STE introduces bias and can lead to optimization instabilities in sequence tasks (e.g., reinforcement learning from human feedback or sequence fine-tuning).

Theoretically, our methods serve as drop-in polynomial surrogates: during training, replace hard decoding with CutMax (for greedy) or nucleus sampling based upon CutMax (for sampling), and compute exact gradients end-to-end. For instance, in a sequence loss $\mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$ where $\hat{\mathbf{y}}$ is generated via decoding, the gradient $\partial\mathcal{L}/\partial\mathbf{x}$ (w.r.t. input logits $\mathbf{x}$) flows precisely through the polynomial chain. While we leave empirical validation (e.g., on BLEU/ROUGE improvements in fine-tuning) to future work, the theoretical differentiability, rooted in smooth plaintext operations and polynomial preservation, positions CutMax as a promising alternative for stable, gradient-based sequence optimization.

4 Empirical Evaluation

To evaluate our methods, we use the `ccdv/arxiv-summarization` (Cohan et al., 2018) dataset of paper abstracts, sampling $N = 100$ articles unless otherwise specified. We report results on realistic LLM logits from models including QWEN2.5-0.5B ($|\mathcal{V}| \approx 150$K), Mistral-7B ($|\mathcal{V}| \approx 33$K), and GPT-2 ($|\mathcal{V}| \approx 50$K).

4.1 Plain-Text CutMax

We evaluate our CutMax in the unencrypted domain on the QWEN2.5-0.5B model (vocabulary size $|\mathcal{V}| \approx 150$K), using the `ccdv/arxiv-summarization` (Cohan et al., 2018) dataset of paper abstracts. We sampled $N = 100$ articles and, for each, retrieved the next-token logits via one-step greedy decoding (*greedy argmax* on the plaintext logits), yielding 100 ground-truth tokens. For each prompt we then ran CutMax with $T \in \{2, 3, 4, 5\}$ iterations, grid-searching over odd powers $p \in \{7, 9, \ldots, 19\}$ and scaling factors $c \in \{3, 5, \ldots, 39\}$ to find the smallest (p, c) that still achieves perfect next-token recovery.

Convergence vs. #iterations. Table 2 summarizes, for each T, the best (p, c) choice, the resulting average maximum normalized score after T iterations, $\text{Argmax}_i = Z_i / \sum_{j=1}^{\mathcal{V}} Z_j$, and the recovery rate (fraction of times Argmax_i^T matches the true token). The results show rapid convergence: even with just $T = 2$ iterations, CutMax concentrates $\approx 95\%$ of the mass on the correct token. By $T = 4$, the recovery exceeds 99.99%, and by $T = 5$, it achieves exact recovery in all cases.

Sparsity and Accuracy. Across all 400 CutMax invocations (100 prompts × 4 values of T), we achieve 100% next-token recovery. Moreover, after convergence (e.g., $T \geq 4$), the distribution is effectively one-hot: out of ≈ 150K vocabulary items, only the true token carries non-negligible mass. These results confirm the theoretical analysis of Sect. 3: CutMax drives a highly skewed distribution in just a handful of iterations, making it an excellent practical choice for homomorphic greedy decoding under CKKS-style encryption.

Table 1: **Homomorphic** argmax **latency and accuracy.** We benchmark CutMax on two realistic LLM logit tensors—Mistral-7B ($|\mathcal{V}| = 32{,}768$) and Qwen-2.5-0.5B ($|\mathcal{V}| = 151{,}936$)—using 100 prompts from the ARXIV-summarisation set. Hyper-parameters are fixed to T=4 with p=[16,4,4,6] and c=[5,3,3,3] due to chain index optimization and numerical stability. We report the mean of the average of maximum values, the average latency, the accuracy of slot predictions, the arrays' sizes, and the device used.

Algorithm	Dataset	Model	Array Length	$\mathbb{E}[\max_i Z_i] \uparrow$	Avg. Latency $\downarrow$ (secs)	Accuracy $\uparrow$	Devices
CutMax	arXiv	Mistral7B-v0.3	32,768	0.994	3.373	100.0%	1× NVIDIA H100
CutMax	arXiv	Qwen2.5	151,936	0.989	4.607	100.0%	1× NVIDIA H100
Baselines							
Nexus	GLUE	BERT	30,522	—	79.36*	—	4× Tesla A100
Nexus	GLUE	Llama-3	128,256	—	162.8*	—	4× Tesla A100

* Nexus (Zhang et al., 2024a) Table 2: "We batched 32 inputs in total ... Runtime is the amortized latency of each input.", originally reported (2.48,5.09) when multiplied by 32 we get (79.36, 162.8), respectively.

Table 2: Convergence of CutMax on QWEN2.5-0.5B ($|\mathcal{V}| \approx 150$K) over 100 arXiv abstracts. We report the mean of the per-prompt Argmax_i^T, the best (i.e. largest) Argmax_i^T observed across the 100 prompts with exact next-token recovery of 100% in all cases.

Iter. (T)	Best (p, c)	$\mathbb{E}[\text{argmax}_i] \uparrow$	$\max(\text{argmax}_i) \uparrow$
2	$(19, 5)$	0.952	0.987
3	$(19, 27)$	0.987	0.994
4	$(13, 15)$	0.999996	1.000
5	$(9, 15)$	1.000	1.000

4.2 HE CutMax

We evaluated our FHE CutMax implementation under the CKKS scheme using HELayers (Aharoni et al., 2023). Experiments were conducted on LLM logits from two models: Qwen-2.5-0.5B and Mistral-7B, with vocabulary sizes of 151,936 and 32,768, respectively. For each model, we used 100 examples and set following hyperparameters: $c = [5, 3, 3, 3]$, $p = [16, 4, 4, 6]$, $T = 4$, chosen to optimize the CKKS chain index and ensure numerical stability. We report the mean of the per-example maximum values, average latency, and slot prediction accuracy. As a baseline, we compare our method against the recently proposed HE-friendly argmax algorithm by Zhang et al.. Results are reported in Table 1, demonstrating that our method is both accurate and efficient. From an accuracy perspective, our method does not alter the model's predictions in any of the tested cases, achieving 100% accuracy. For efficiency benchmarking, our method significantly reduces the latency of argmax compared to Nexus (Zhang et al., 2024a), by several orders of magnitude for comparable vocabulary sizes. Specifically, for a vocabulary size of approximately 30K, our method reduces latency from 79.56 seconds to 3.373 seconds. For a larger vocabulary of approximately 150K, our method reduces latency from 162.8 to 4.607. Note that we chose Nexus as our baseline even though there are newer papers on LLM inference such as (Zimerman et al., 2024a; Park et al., 2025; Zhang et al., 2025b; Moon et al., 2024b; Kei and Chow, 2025), because these did not include an argmax benchmark or did not utilize argmax at all.

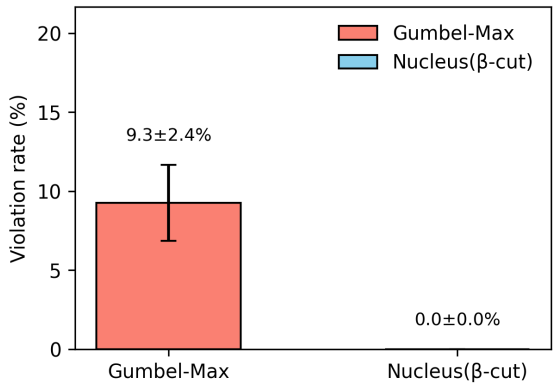


Figure 4: **Violation Rates in Nucleus Sampling Methods**: Violation rate (%) of sampling outside the top-p nucleus for Gumbel-Max and Nucleus(β-cut), averaged over 100 prompts with 1000 draws each ($p = 0.9$). Error bars denote one standard deviation.

4.3 Nucleus Sampling

Figure 5 qualitatively demonstrates our method's effectiveness, showing a sharply truncated distribution that confirms all probability mass remains within the top-p nucleus. To verify that our one-shot nucleus sampler never selects tokens outside the intended top-p set, we measured the *violation rate* as the fraction of draws falling outside the nu-

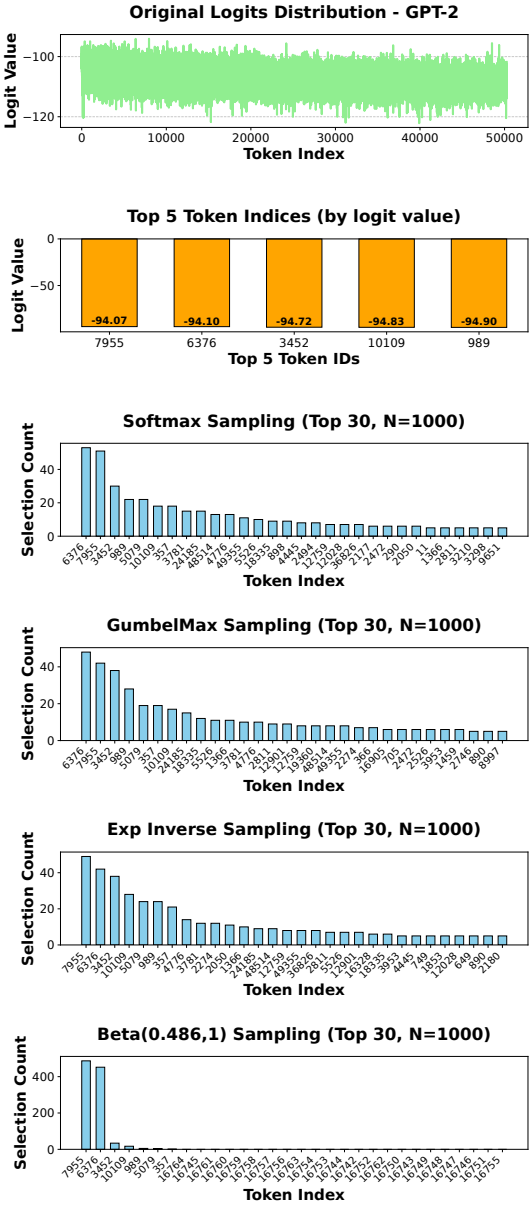


Figure 5: **Comparison of Sampling Methods on GPT-2 Last-Token Decoding:** The top strip shows the complete 50,257-dimensional logit vector (only its outline is visible at this scale), and the second strip zooms in on the five largest logits. The three lower strips plot, for 1,000 repeated draws, histograms of the 30 most frequently chosen token indices produced by four different samplers: standard Softmax at temperature 1, the classical Gumbel-Max trick, inverse-transform sampling on the normalized exponentials ("Exp-inverse"), and our one-shot *Beta-cut nucleus* sampler, which perturbs each logit with i.i.d. $G_i \sim \mathrm{Beta}(\alpha, 1)$ noise using $\alpha = 0.486$ computed from Theorem 3.4 for $(p, q) = (0.9, 0.95)$. Whereas the first three methods still allocate non-negligible mass to tail tokens, the Beta-cut histogram is sharply truncated, confirming that all probability remains inside the top-p nucleus exactly as required by the HE-friendly design of Sect. 3.3.

cleus, over $S = 100$ prompts, with $N = 1000$ samples per prompt and $p = 0.9$. We compare standard Gumbel-Max sampling against our Nucleus(β–cut) method. Results are summarized in Figure 4: Gumbel-Max violates the top-p constraint in $9.3\% \pm 2.4\%$ of draws, whereas Nucleus (β-cut) has *zero* violations.

5 Conclusions

We solve a critical bottleneck in privacy-preserving AI: adapting LLM decoding for homomorphic encryption, where non-polynomial operations like argmax and sampling are computationally prohibitive. We introduce **CutMax**, a novel, HE-friendly and theoretically grounded argmax algorithm that replaces slow, comparison-based methods with an efficient iterative polynomial process. Crucially, CutMax achieves 100% accuracy and executes in just a few seconds on large vocabularies ($|\mathcal{V}| \approx 150$K), making secure greedy decoding practical for LLM inference for the first time. Building on this, we present the **first HE-compatible nucleus (top-p) sampling method**, which leverages CutMax with the idea of noisy inverse transform sampling to provably restrict sampling to the desired token set. Together, these fully polynomial algorithms provide a complete and efficient framework for both greedy and stochastic decoding over encrypted data, addressing a major barrier to the deployment of secure LLMs in real-world applications.

Limitations

Our work focuses on optimizing the decoding stage of encrypted token generation. While this is a critical component, it does not capture the full computational cost of encrypted inference. We view this work as a step toward narrowing the still-substantial gap between encrypted and plaintext generation, and we believe that combining our techniques with future advances, such as attention mechanisms, can help enable practical, privacy-preserving generative AI. Our implementation uses the CKKS scheme, which is well-suited for approximate arithmetic and widely adopted in encrypted machine learning. While the approach is conceptually compatible with other HE schemes, extending it may require engineering adaptations. As the field evolves, e.g., through new HE schemes or hardware support, our techniques can be adapted accordingly. Our paper focuses on improving the efficiency of LLM

decoding methods under encryption. Therefore, it preserves the behavior of the underlying generative model, including any biases or factual inaccuracies it may contain. While encryption protects user data, it also limits visibility into model behavior, making post-hoc filtering and auditing more challenging. We view this as a crucial direction for future research in secure and responsible AI.

Ethical Considerations

Our work aims to enhance privacy in text generation by enabling efficient LLM decoding on encrypted inputs. This can reduce the exposure of sensitive user data when interacting with large language models. However, our method preserves the behavior of the underlying model, including any biases, inaccuracies, or harmful content it may produce. Moreover, since encrypted inference limits visibility into intermediate states, it may complicate post-generation auditing and filtering. We view secure model auditing and bias mitigation as important complementary directions for future research.

References

Allon Adir, Ehud Aharoni, Nir Drucker, Ronen Levy, Hayim Shaul, and Omri Soceanu. 2024. *Homomorphic Encryption for Data Science (HE4DS)*. Springer.

Ehud Aharoni, Allon Adir, Moran Baruch, Nir Drucker, Gilad Ezov, Ariel Farkash, Lev Greenberg, Ramy Masalha, Guy Moshkowich, Dov Murik, et al. 2023. HElayers: A tile tensors framework for large neural networks on encrypted data. *Proceedings on Privacy Enhancing Technologies (PoPETs)*.

Moran Baruch, Nir Drucker, Gilad Ezov, Yoav Goldberg, Eyal Kushnir, Jenny Lerner, Omri Soceanu, and Itamar Zimerman. 2024. Polynomial adaptation of large-scale CNNs for homomorphic encryption-based secure inference. In *International Symposium on Cyber Security, Cryptology, and Machine Learning (CSCML)*, pages 3–25. Springer.

Moran Baruch, Nir Drucker, Lev Greenberg, and Guy Moshkowich. 2022. A methodology for training homomorphic encryption friendly neural networks. In *International Conference on Applied Cryptography and Network Security (ACNS)*, pages 536–553. Springer.

Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. 2013. Estimating or propagating gradients through stochastic neurons for conditional computation. *ArXiv*, abs/1308.3432.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, et al. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems (NeurIPS)*.

Olive Chakraborty and Martin Zuber. 2022. Efficient and accurate homomorphic comparisons. In *Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, WAHC'22, page 35–46, New York, NY, USA. Association for Computing Machinery.

Jung Hee Cheon, Andrey Kim, Miran Kim, and Yongsoo Song. 2017. Homomorphic encryption for arithmetic of approximate numbers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 409–437. Springer.

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*, pages 415–445, Cham. Springer International Publishing.

Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2016. Faster fully homomorphic encryption: Bootstrapping in less than 0.1 seconds. In *Advances in Cryptology – ASIACRYPT 2016*, pages 3–33, Berlin, Heidelberg. Springer Berlin Heidelberg.

Arman Cohan, Franck Dernoncourt, Doo Soon Kim, Trung Bui, Seokhwan Kim, Walter Chang, and Nazli Goharian. 2018. A discourse-aware attention model for abstractive summarization of long documents. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 615–621, New Orleans, Louisiana. Association for Computational Linguistics.

Jack L. H. Crawford, Craig Gentry, Shai Halevi, Daniel Platt, and Victor Shoup. 2018. Doing real work with fhe: The case of logistic regression. In *Proceedings of the 6th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, WAHC '18, page 1–12, New York, NY, USA. Association for Computing Machinery.

Leo de Castro, Antigoni Polychroniadou, and Daniel Escudero. 2024. Privacy-preserving large language model inference via GPU-accelerated fully homomorphic encryption. In *Neurips Safe Generative AI Workshop 2024*.

Angela Fan, Mike Lewis, and Yann Dauphin. 2018. Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 889–898, Melbourne, Australia. Association for Computational Linguistics.

Lars Folkerts and Nektarios Georgios Tsoutsos. 2024. Tyche: Probabilistic selection over encrypted data for generative language models. Cryptology ePrint Archive, Paper 2024/1087.

Craig Gentry. 2009. *A fully homomorphic encryption scheme*. Ph.D. thesis, Stanford University, Palo Alto, CA.

Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 201–210, New York, New York, USA. PMLR.

Robert E Goldschmidt. 1964. *Applications of division by convergence*. Ph.D. thesis, Massachusetts Institute of Technology.

Arnaud Grivet Sébert, Rafael Pinot, Martin Zuber, Cédric Gouy-Pailler, and Renaud Sirdey. 2021. Speed: secure, private, and efficient deep learning. *Machine Learning*, 110(4):675–694.

Emil Julius Gumbel. 1954. *Statistical theory of extreme values and some practical applications: a series of lectures*, volume 33. US Government Printing Office.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. 2019a. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*.

Ari Holtzman, Jan Buys, Maxwell Forbes, and Yejin Choi. 2019b. The curious case of neural text degeneration. *CoRR*, abs/1904.09751.

Ilia Iliashenko and Vincent Zucca. 2021. Faster homomorphic comparison operations for bgv and bfv. *Proceedings on Privacy Enhancing Technologies (PoPETs)*, 2021:246 – 264.

Nikola Jovanovic, Marc Fischer, Samuel Steffen, and Martin Vechev. 2022. Private and reliable neural network inference. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, CCS '22, page 1663–1677, New York, NY, USA. Association for Computing Machinery.

Andes Y. L. Kei and Sherman S. M. Chow. 2025. SHAFT: Secure, Handy, Accurate, and Fast Transformer Inference. In *Network and Distributed System Security Symposium (NDSS)*, volume 2025. Internet Society.

Chris J. Maddison, Daniel Tarlow, and Tom Minka. 2014. A* sampling. In *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.

Federico Mazzone, Maarten Everts, Florian Hahn, and Andreas Peter. 2025. Efficient ranking, order statistics, and sorting under ckks.

Jungho Moon, Zhanibek Omarov, Donghoon Yoo, Yongdae An, and Heewon Chung. 2024a. Adaptive successive over-relaxation method for a faster iterative approximation of homomorphic operations. Cryptology ePrint Archive, Paper 2024/1366.

Jungho Moon, Dongwoo Yoo, Xiaoqian Jiang, and Miran Kim. 2024b. THOR: Secure transformer inference with homomorphic encryption. Cryptology ePrint Archive, Paper 2024/1881.

Samanvaya Panda. 2021. Principal component analysis using ckks homomorphic scheme. In *Cyber Security Cryptography and Machine Learning*, pages 52–70, Cham. Springer International Publishing.

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.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners.

Biwei Yan, Kun Li, Minghui Xu, Yueyan Dong, Yue Zhang, Zhaochun Ren, and Xiuzhen Cheng. 2024. On protecting the data privacy of large language models (llms): A survey. *arXiv preprint arXiv:2403.05156*.

Yifan Yao, Jinhao Duan, Kaidi Xu, Yuanfang Cai, Zhibo Sun, and Yue Zhang. 2024. A survey on large language model (llm) security and privacy: The good, the bad, and the ugly. *High-Confidence Computing*, 4(2):100211.

Jiawen Zhang, Kejia Chen, Zunlei Feng, Jian Lou, Mingli Song, Jian Liu, and Xiaohu Yang. 2025a. Secpe: Secure prompt ensembling for private and robust large language models.

Jiawen Zhang, Jian Liu, Xinpeng Yang, Yinghao Wang, Kejia Chen, Xiaoyang Hou, Kui Ren, and Xiaohu Yang. Secure transformer inference made non-interactive (2023). *URL https://eprint. iacr. org/2024/136. Publication info*, 1:1–1.

Jiawen Zhang, Xinpeng Yang, Lipeng He, Kejia Chen, Wen jie Lu, Yinghao Wang, Xiaoyang Hou, Jian Liu, Kui Ren, and Xiaohu Yang. 2024a. 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. 2025b. MOAI: Module-optimizing architecture for non-interactive secure transformer inference. Cryptology ePrint Archive, Paper 2025/991.

Peng Zhang, Ao Duan, and Hengrui Lu. 2024b. An efficient homomorphic argmax approximation for privacy-preserving neural networks. *Cryptography*, 8(2).

Itamar Zimerman, Allon Adir, Ehud Aharoni, Matan Avitan, Moran Baruch, Nir Drucker, Jenny Lerner, Ramy Masalha, Reut Meiri, and Omri Soceanu. 2024a. Power-softmax: Towards secure llm inference over encrypted data. *arXiv preprint arXiv:2410.09457*.

Itamar Zimerman, Moran Baruch, Nir Drucker, Gilad Ezov, Omri Soceanu, and Lior Wolf. 2024b. Converting transformers to polynomial form for secure inference over homomorphic encryption. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org.

Appendix

A Threat model

There are two commonly used threat-models for secure inference and generative AI for LLMs over HE: 1) using encrypted weights; or 2) using encrypted input samples (queries). When considering a 2-party scenario with HE, it is commonly assumed that one party is the (untrusted, semi-honest) server who holds the model (encrypted or not) and the other party is the data-owner who performs the query and would like to avoid revealing the query to the server. The decision of whether the server holds the model encrypted or not depends on whether a third-party the model-owner agrees to share the data with the server. Our proposed solution is orthogonal to the above decision, which eventually only affects latency. All communications are performed using TLS 1.3 and privacy attacks such as model extraction attack and membership inference attacks require an extra mechanism such as differential privacy, which is also orthogonal to this work. For brevity, we have decided to refer the reader to (Adir et al., 2024)[Chapter 3] that further describes this security model.

B Approximations for HE

Since HE supports only polynomial operations, functions like $\max$, SIGN, Inv, and InvSqr must be approximated. A common formulation for $\max(a, b)$ is:

$$\max(a, b) = \frac{a + b}{2} + \frac{\text{SIGN}(a - b) \cdot (a - b)}{2}, \tag{5}$$

where $\text{SIGN}(x) = 1$ when $x > 0$ and -1 otherwise. An approximation of SIGN is given, e.g., in (Moon et al., 2024a). To make the results of SIGN suitable as an indicator function (as in Equation (5)), it is common to translate the SIGN range from $\{-1, 1\}$ to $\{0, 1\}$ using $\frac{1+\text{SIGN}(x)}{2}$. The $\max$ and SIGN functions are used by prior-art argmax implementations. In contrast, CutMax uses Inv and InvSqr approximations.

B.1 Goldschmidt

The *Goldschmidt* inverse algorithm (Goldschmidt, 1964) is an iterative process for computing the function $f(x) = \frac{1}{x}$. (Cheon et al., 2019) showed that this algorithm can be used to approximate inputs in the range $[0, 2]$ effectively using the equation

$$f(x) = \prod_{i=0}^{d} \left(1 + (1 - x)^{2^i}\right) \tag{6}$$

Specifically, they proved that for $(x \in \left[\frac{1}{2^n}, 1\right])$, it requires $(d = \Theta(\log \alpha + n))$ iterations to converge, with an error bound of (2^{α}). (Panda, 2021) showed an efficient implementation for the Goldschmidt inverse square root (InvSqr) algorithm over HE. Subsequently, (Moon et al., 2024a) reported improved measurements of the inverse and InvSqr implementations under CKKS. Table 3 repeats some benchmarks for the SIGN and InvSqr functions at different approximation levels (α). One immediate conclusion is that the InvSqr implementation is $(1.5 - 2.1\times)$ faster compared to the SIGN implementation for the same level of errors. One reason is that InvSqr has a lower *multiplication depth* (defined below) and thus requires fewer bootstraps.

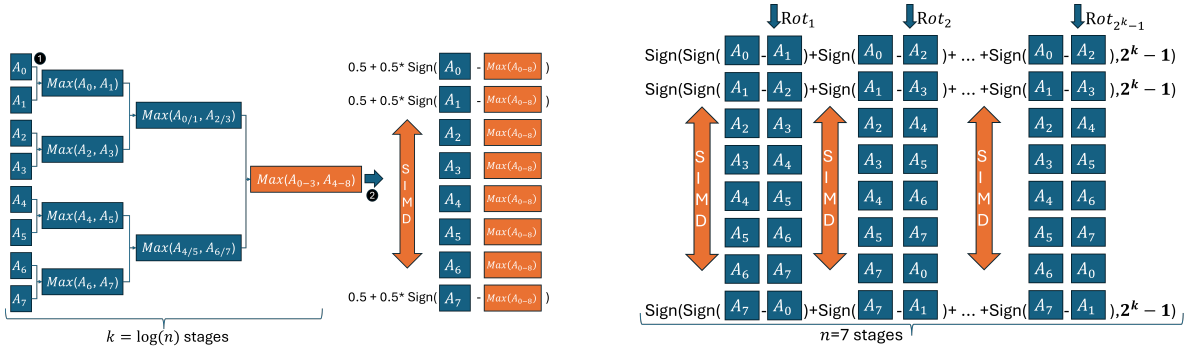
Multiplication Depth. In CKKS, a ciphertext must be refreshed via a costly *bootstrap* operation after every L sequential multiplications, where L is a scheme parameter. Since bootstrapping is significantly slower than other operations, a key optimization goal is to minimize its use. The longest chain of multiplications in a function—its *multiplication depth*, is therefore a critical target for reduction.

C Previous, Comparison-Based Methods for argmax over HE

Figure 6 visualizes the two canonical comparison-based approaches for encrypted argmax: the *tournament* method (Figure 6a) and the *league* method (Figure 6b). Both rely on repeated encrypted comparisons implemented via polynomial approximations of $\text{SIGN}(\cdot)$.

Table 3: A sample benchmarks of (Moon et al., 2024a) conducted on AMD EPYC 7502-32 CPU, using Microsoft Seal, poly degree of 2^{17}, and scale $\Delta = 2^{40}$.

Algorithm	α	7	9	11	13
SIGN	Iterations	7	9	11	12
	Multiplication depth	14	18	22	24
	Time (s)	6.3	11.7	20.2	26.9
InvSqr	Iterations	5	6	7	8
	Multiplication depth	10	12	14	16
	Time (s)	4.1	6.6	9.8	12.8



(a) Tournament method.

(b) League method.

Figure 6: **Illustration of Prior Comparison-based argmax Methods in HE** (see main text Sect. 2).

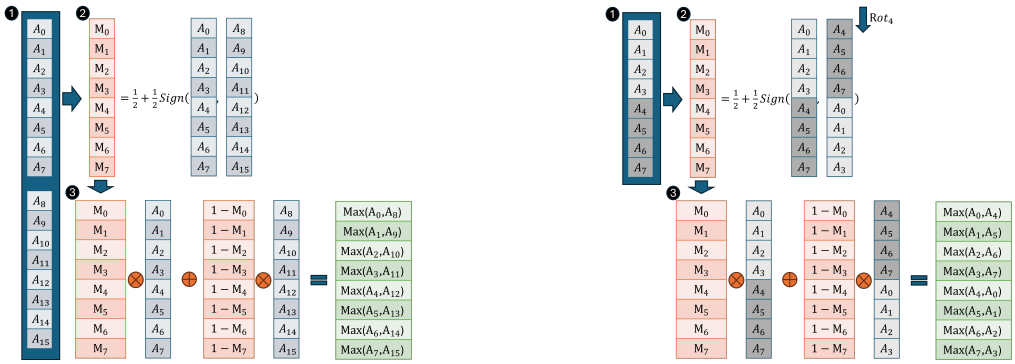
In the tournament design, approximately $n-1$ pairwise comparisons are arranged into $\log_2 n$ sequential stages. At each stage, the number of candidates is halved until only the maximum remains. While asymptotically efficient, this design requires $\log n$ sequential SIGN evaluations, each incurring costly rotations and bootstraps.

In the league design, every pair of inputs is compared, and results are accumulated so that the true maximum is the only element with score $n-1$. This requires n sequential rounds of comparisons. Although SIMD parallelism allows all n pairwise comparisons within a round to be executed in parallel slots, the *sequential* round structure still dominates, leaving the method quadratic overall.

As summarized in Table 3, polynomial SIGN approximations are substantially deeper and slower than inverse-square-root or division. Thus, even when accelerated with SIMD, both the tournament and league methods remain inefficient at large n. In practice, this has limited published HE argmax implementations either to small input sizes or to systems that decrypt logits before argmax.

D Argmax using SIMD

SIMD and Rotations. CKKS enables SIMD-style packing, where one ciphertext encrypts a vector $(v_1, \ldots, v_s)$. Operations are applied elementwise, and encrypted vectors can be rotated by $\ell < s$ positions. This allows for patterns such as RotAndSum (Adir et al., 2024) - which aggregates values with $\log s$ rotations and additions. Figure 7 demonstrates how to leverage SIMD to execute n $\max(a, b)$ operations in parallel. Step (1) shows the encrypted input; in Step (2), we compute an indicator vector where $M[i] = 1$ if $A[i] > A[i + \frac{n}{2}]$, and 0 otherwise. Subsequently, Step (3) selects (masks) only the elements that match the indicator vector, which returns the desired results. Consider two cases: $n > s$ (Figure 7a) and $n = s$ (Figure 7b) (The case $n < s$ behaves similarly to $n = s$). Here, s is the maximal number of elements that a single ciphertext can encrypt. Figure 7a illustrates the case where $n = 16$ elements are encrypted in two ciphertexts, each containing a vector of 8 elements. In contrast, Figure 7b shows the case $n = s$, which requires only one ciphertext. In the first case, no rotation is needed, as we can directly subtract one ciphertext from the other using element-wise subtraction. However, this is not the case in Figure 7b, where we must homomorphically rotate A by $n/2$ positions. As a result, the output



(a) $n > s$. (b) $n = s$.

Figure 7: HE SIMD example implementation.

is duplicated, and we only need to keep the top $n/2$ elements. It appears that many comparisons can be saved by using SIMD. Consider the case where $n = s$ for the league method (Figure 6b) that requires only n homomorphic rotations and n parallel SIGN evaluations, with one additional final parallel SIGN evaluation. The complexity is thus reduced from n^2 to n SIGN operations, which, on hardware with sufficient parallel threads, may result in faster execution than the sequential tournament algorithm. When $n > s$ it is worth combining the methods as was done in (Iliashenko and Zucca, 2021). Starting with a tournament and when the number of elements is reduced beyond some threshold continue with the league method.

E Convergence of CutMax

E.1 Preliminaries and Definitions

The following definitions assume that a vector $x \in \mathbb{R}^n$ has a unique maximum element. We also denote the average of x by $\mu_x = \frac{1}{n}\sum_i x_i$ and its standard deviation by $\sigma_x = \sqrt{\frac{1}{n}\sum_i (x_i - \mu_x)^2}$.

Definition E.1 (Max elements)**.** *Given a vector $x \in \mathbb{R}^n$ with a unique maximal element, the index of that element is denoted by $i_x^* = \operatorname{argmax}_i x_i$ and the index of the second-largest element (excluding the maximum) is denoted by $i_x^{**} = \operatorname{argmax}_{j \neq i_x^*} x_j$.*

Definition E.2 (δ-gap bound)**.** *A vector $x \in \mathbb{R}^n$ with a unique maximum at index i_x^* is δ-gap bounded for $\delta \in \mathbb{R}_+$ when $x_{i_x^*} - x_{i_x^{**}} \geq \delta \cdot \sigma_x$*

Definition E.3 (Gap Ratio)**.** *For $y \in \mathbb{R}^n$, its* gap ratio *is $\Gamma(y) = \frac{y_{i_y^*}}{y_{i_y^{**}}}$*

Definition E.4 (c-semi-standardization)**.** *For $c \in \mathbb{R}_+$, a c-semi-standardization function is*

$$S_c(y) : \mathbb{R}^n \longrightarrow \mathbb{R}^n \tag{7}$$

$$y_i \longmapsto \frac{y_i - \mu_y}{c \cdot \sigma_y} + 1 \tag{8}$$

Definition E.5 (CutMax Iteration)**.** *Given $c \in \mathbb{R}_+$ and an odd $p \in \mathbb{Z}_+$, the* CutMax *iteration function is*

$$\mathrm{CutMax}_t(y^{(t)}) : \mathbb{R}^n \longrightarrow \mathbb{R}^n \tag{9}$$

$$y_i^{(t)} \longmapsto \left(S_c(y^{(t)})\right)^p \tag{10}$$

for some iteration index $t > 0$.

E.2 Key Lemmas

The following lemma shows that running a standardization on a vector y preserves the ordering of elements in it. In addition, it transforms the gap condition.

Lemma E.6. *The c-semi-standardization function (S_c) is a strictly increasing function.*

Proof. For every $y \in \mathbb{R}^n$ and for every $i, j \in [n], i \neq j$, because $\mu_y, \sigma_y, c > 0$ it follows that

$$y_i > y_j \iff y_i - \mu_y > y_j - \mu_y \tag{11}$$

$$\iff \frac{y_i - \mu_y}{c \cdot \sigma_y} > \frac{y_j - \mu_y}{c \cdot \sigma_y} \iff S_c(y)_i > S_c(y)_j \tag{12}$$

□

Lemma E.7. *Let $y \in \mathbb{R}^n$ be a δ-gap bounded vector then $S_c(y)$ is a $\frac{\delta}{c}$-gap bounded vector.*

Proof. Set $z := S_c(y)$, $i_z^* = i_y^*$ and $i_z^{**} = i_y^{**}$. then

$$z_{i_z^*} - z_{i_z^{**}} = \frac{y_{i_y^*} - y_{i_y^{**}}}{c \cdot \sigma_y} \underset{y \text{ is } \delta\text{-gap bounded}}{\geq} \frac{\delta \cdot \sigma_y}{c \cdot \sigma_y} = \frac{\delta}{c} \tag{13}$$

□

The next Lemma shows that applying a c-semi-standardization function S_c on some vector y results with negative coefficients in places where y_i is at least c standard deviations below the mean of y.

Lemma E.8 (Unconditional gap growth for one CutMax step)**.** *Let $Y^{(t)} \in \mathbb{R}^n$ have a unique maximizer at i^* and let $i^{**} \neq i^*$ be the runner-up index. Assume $Y^{(t)}$ is δ-gap bounded in the sense that $Y_{i^*}^{(t)} - Y_{i^{**}}^{(t)} \geq \delta\, \sigma_{Y^{(t)}}$ (Def. E.2), where $\sigma_{Y^{(t)}}$ denotes the population standard deviation. Assume additionally that $\min_i y_i^{(t)} \geq \mu_{y^{(t)}} - c\sigma_{y^{(t)}} + \epsilon$ for small $\epsilon > 0$, ensuring all $z_i > 0$. Fix $c > 0$ and an odd integer $p \geq 1$, and set $z = S_c(Y^{(t)}) = \frac{Y^{(t)} - \mu_{Y^{(t)}}}{c\,\sigma_{Y^{(t)}}} + 1$ and $\mathbf{Y}^{(t+1)} = z^{\odot p}$ (the elementwise odd-power step of CutMax). Then the post-step gap ratio satisfies $\frac{\mathbf{Y}_{i^*}^{(t+1)}}{\mathbf{Y}_{i^{**}}^{(t+1)}} = \left(\frac{z_{i^*}}{z_{i^{**}}}\right)^p \geq \left(1 + \frac{\delta}{c+\sqrt{n}}\right)^p$. Consequently, after T CutMax iterations, $\frac{y_{i^*}^{(T)}}{y_{i^{**}}^{(T)}} \geq \left(1 + \frac{\delta}{c+\sqrt{n}}\right)^{pT}$.*

Proof. By Lemma E.7, $z_{i^*} - z_{i^{**}} \geq \delta/c$. Therefore

$$\frac{z_{i^*}}{z_{i^{**}}} = 1 + \frac{z_{i^*} - z_{i^{**}}}{z_{i^{**}}} \;\geq\; 1 + \frac{\delta}{c\, z_{i^{**}}}.$$

To upper-bound $z_{i^{**}}$, write $r_i = \frac{Y_i^{(t)} - \mu_{Y^{(t)}}}{\sigma_{Y^{(t)}}}$. From the definition of the population standard deviation, we have $\frac{1}{n}\sum_i r_i^2 = 1 \Rightarrow \sum_i r_i^2 = n$, hence $r_{i^{**}} \leq \sqrt{n}$ and thus $z_{i^{**}} = 1 + \frac{r_{i^{**}}}{c} \leq 1 + \frac{\sqrt{n}}{c}$. Combining the bounds gives

$$\frac{z_{i^*}}{z_{i^{**}}} \;\geq\; 1 + \frac{\delta}{c\,(1 + \sqrt{n}/c)} = 1 + \frac{\delta}{c + \sqrt{n}}.$$

Finally, the CutMax power step yields $\frac{\mathbf{Y}_{i^*}^{(t+1)}}{\mathbf{Y}_{i^{**}}^{(t+1)}} = \left(\frac{z_{i^*}}{z_{i^{**}}}\right)^p$, which gives the stated inequality. Iterating T times multiplies the exponent by T. □

Lemma E.9 (Per-step gap amplification)**.** *Let $\mathbf{y}^{(t)} \in \mathbb{R}^n$ satisfy the δ–gap condition* (Def. E.2)*, fix $c > 0$ and odd $p \geq 1$, and set $z = S_c(\mathbf{y}^{(t)})$ and $\mathbf{y}^{(t+1)} = z^{\odot p}$. Then the gap ratio satisfies*

$$\Gamma^{(t+1)} \;=\; \Gamma^{(t)} \left(\frac{z_{i^*}}{z_{i^{**}}}\right)^p \;\geq\; \Gamma^{(t)} \left(1 + \frac{\delta}{c + \sqrt{n}}\right)^p.$$

Proof. By Lemma E.8, $\frac{z_{i^*}}{z_{i^{**}}} \geq 1 + \frac{\delta}{c+\sqrt{n}}$. Since $\Gamma^{(t+1)} = \Gamma^{(t)}(z_{i^*}/z_{i^{**}})^p$, the claim follows. □

Theorem E.10 (CutMax Convergence)**.** *Let $\mathbf{x} \in \mathbb{R}^n$ satisfy the δ–gap condition with $\delta > 0$, and run CutMax with parameters (p, c, T), $c > 0$, odd $p \geq 1$. Then*

$$\Gamma^{(T)} \;\geq\; \Gamma^{(0)} \left(1 + \frac{\delta}{c + \sqrt{n}}\right)^{pT}.$$

In particular, to ensure $\Gamma^{(T)} \geq k$, it suffices to take

$$T \;\geq\; \left\lceil \frac{\ln k - \ln \Gamma^{(0)}}{p\, \ln\!\left(1 + \frac{\delta}{c+\sqrt{n}}\right)} \right\rceil \;\leq\; \left\lceil \frac{\ln k}{p\, \ln\!\left(1 + \frac{\delta}{c+\sqrt{n}}\right)} \right\rceil,$$

where the last inequality uses $\Gamma^{(0)} \geq 1$.

Proof. Iterating Lemma E.9 yields $\Gamma^{(T)} \geq \Gamma^{(0)} \left(1 + \frac{\delta}{c+\sqrt{n}}\right)^{pT}$. Solving for T gives the stated bound. □

Table 4: Min. iterations $T_{\min}$ for $n = 32{,}768$ (Mistral-7B), $p = 13$, $c = 5$.

δ	$k = 10$	$k = 100$
0.5	66	132
1	34	67
2	17	34
3	12	23
5	7	14
10	4	7