Tyche: Probabilistic Selection over Encrypted Data For Generative Language Models

Lars Folkerts and Nektarios Georgios Tsoutsos

University of Delaware, Newark, DE, USA
folkerts@udel.edu tsoutsos@udel.edu

Abstract. Generative AI, a significant technological disruptor in recent years, has impacted domains like augmented reality, coding assistance, and text generation. However, use of these models requires users to trust the model owners with their sensitive data given as input to the model. Fully Homomorphic Encryption (FHE) offers a promising solution, and many earlier works have investigated the use this technology for machine learning as a service (MLaaS) applications. Still, these efforts do not cater to generative models which operate probabilistically, allowing for diverse and creative outputs. In this work, we introduce three novel probabilistic selection algorithms for autoregressive generative AI: multiplication-scaled cumulative sum, heuristic cumulative sum, and the random-multiplication argmax. Each of these approaches presents distinctive challenges in optimizing the trade-off between precision and timing performance, a balance intricately tied to the specific characteristics of the data under consideration. Our results show that the random multiplication argmax-based method is more scalable than the cumulative sum methods and can accurately mimic the plaintext selection curve.

Keywords: Fully Homomorphic Encryption · Private Language Models · Generative AI.

1 Introduction

Over the past decade, generative AI has become a significant disruptor for modern technology, offering automated solutions for content generation and data processing. Many utilize this technology for increased efficiency and productivity in their day-to-day work, and new professions have been created for those skilled at using these models. It has gained widespread recognition and piqued considerable interest as a foundational technology in today's rapidly evolving landscape [25, 47, 49].

Generative AI operates probabilistically, employing statistical models to generate new data that loosely resemble a given training dataset. At its core, an autoregressive generative model functions by iteratively generating output tokens one at a time [14]. Given an input prompt, the model generates a set of probabilities corresponding to the next possible output token. It then randomly selects the next token based on these probabilities and appends it to the input

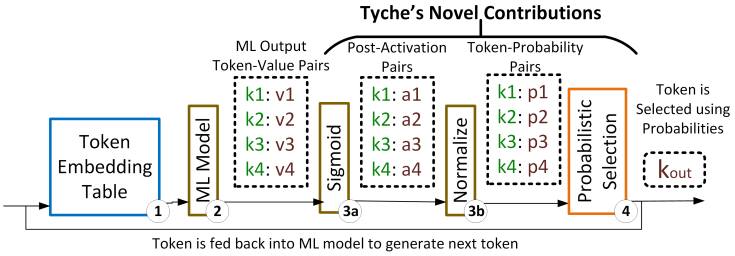


Fig. 1: **Generative Language Models:** In generative models, the process of selecting the most likely next token can be simplified into the four steps shown in this figure. Our research primarily addresses the latter two phases: Final Activation/Normalization and Probabilistic Selection. A discussion on how generative models work is presented in Section 2.1, and the associated steps are numbered in this figure.

for the subsequent iteration. The generative AI model gradually constructs a coherent and realistic output sequence by repeating this iterative process.

The inherent probabilistic nature of generative AI is a key factor contributing to its ability to produce diverse and creative outputs. The randomness introduced during the token selection stage ensures that the model does not adhere strictly to a single predefined path. Instead, it explores different possibilities within the learned probability distributions, allowing for the generation of novel and imaginative content, leading to diverse applications [55, 49].

Nevertheless, the privacy of generative AI systems remains a significant barrier to their widespread adoption. For example, Deutsche Bank stated "On ChatGPT specifically, like other banks we have currently blocked the website while we evaluate how to best use these types of capabilities while also ensuring the security of our and our client's data. So it is protection against data leakage, rather than a view on how useful the tool is" [15]. Several prominent companies, including Amazon, Apple, Northrop Grumman, Samsung, Verizon, and various major banks, have banned the use of ChatGPT due to concerns regarding the security and confidentiality of code and customer data [15, 56, 26].

Fully Homomorphic Encryption (FHE) offers a promising solution for preserving privacy in machine learning applications through its ability to perform computations directly on encrypted data. This exceptional capability allows users to securely delegate computations to a cloud service provider by sending encrypted ciphertexts, which prevents the provider from accessing any information about the original plaintext. The cloud performs the computations on the encrypted data and returns the encrypted result to the users, who can ultimately decrypt it to obtain the plaintext outcome.

Exciting progress is being made in the realm of FHE for Machine Learning as a Service (MLaaS) scenarios, particularly toward lowering the high computational costs. Notably, within the past five years, the state-of-the-art AlexNet architecture for ImageNet [31] classification has seen a substantial improvement in computation time, reducing from an extrapolated period of over two years in

2019 [38] to a matter of hours by 2023 [1, 19]. These costs have been thanks to ongoing algorithmic enhancements and optimized software libraries.

Nevertheless, when it comes to building FHE-GPTs, several open challenges remain. A recent article by Zama AI estimated that the cost of generating a single output token (such as a word) using Chat-GPT homomorphically would amount to approximately $5000 USD [24]. While this prediction poses major limitations, there is optimism that advancements in several key areas will contribute to cost reduction. Continued compression of large language models (LLMs), ongoing improvements in FHE algorithms, and the emergence of dedicated FHE hardware promise to drive down these costs substantially.

This work marks an initial stride towards realizing practical generative machine learning models by assessing multiple techniques for cloud-based encrypted value selection from a set of neural network outputs. This endeavor tackles two unresolved challenges in the realm of machine learning: (a) the normalization of neural network outputs and (b) the retrieval of probabilistic information. The first challenge entails finding an efficient encrypted counterpart to the softmax function, facilitating the normalization of inputs into probability distributions that sum to 1. The second challenge involves probabilistic information retrieval, where given probabilities p_1 to p_n, $token_1$ is selected p_1 percent of the time, and so forth. Finally, we combine our contributions with several multi-layer perception-based networks to demonstrate the possibility of achieving encrypted generative AI.

Our contributions extend to a diverse set of innovative methodologies aimed at enhancing the probabilistic selection process within encrypted generative machine learning. For each of these techniques, we conduct a comprehensive analysis to evaluate their scalability with respect to algorithmic complexity and the required integer precision, a significant factor influencing the timing performance of FHE. Furthermore, we assess the potential bias introduced from precision limitations and optimizations in FHE. To validate the efficacy of these approaches, we perform empirical evaluations employing text-based generative models that operate on letters. These evaluations encompass measurements of timing and the assessment of loss degradation across various character sets.

2 Background

2.1 Generative AI

At the core of generative ML is the concept of probability. The goal is to model the probability distribution of the training data, allowing the algorithm to generate new samples that are likely to occur in the real world. These models can be used to create realistic images, text, audio, and even entire virtual environments. There are several techniques and architectures used in generative ML, each with its own strengths and limitations. Some of the popular approaches include Variational Autoencoders (VAEs), Generative Adversarial Networks (GANs), and Autoregressive models [14].

In this work, we focus on autoregressive generative inference, which is widely used in various applications, including text generation, image synthesis, handwriting generation, and speech synthesis. It encompasses various approaches such as Google's PixelCNN [44] for images, as well as Bidirectional Encoder Representations from Transformers (BERT) [16] and Generative Pretrained Transformers (GPTs) [48] for large language models. Autoregressive models involve utilizing neural networks to predict the next token (such as a pixel, word, or letter) based on previous inputs. The autoregressive approach leverages the sequential nature of the data, allowing the neural network model to capture dependencies and generate coherent and contextually appropriate predictions. For example, in the case of language models like BERT and GPT, the model predicts the next word in a sentence based on the preceding words, taking into account the syntactic and semantic context. Similarly, in PixelCNN, the model predicts the next pixel in an image based on the previously generated pixels, capturing local patterns and structures. The inference server can then select the token based on these probabilities and iterate the process by feeding the selected token back into the model for generating subsequent predictions.

We show an overview of generative language models in Figure 1. There are four najor steps:

1. Token Embedding: Initially, a token embedding table is employed to transform a one-hot vocabulary vector into a lower-dimensional token embedding representation.
2. ML Model Processing: In the second step, the data is processed using a machine learning model, which could be a transformer-based architecture or a simpler neural network, depending on the application. Here we assume the output of this step is encoded as a vector v, with each value corresponding to a unique output token k.
3. Final Activation/Normalization: The third step involves converting the neural network's output (denoted as v), into an intermediate post-activation values (donated as a), and finally into a normalized probability vector (denoted as p). This is achieved by first applying an activation function, before ensuring that the sum of the outputs equals one, creating a valid probability distribution. In the plaintext domain, softmax is typically used for this step, which combines the sigmoid activation and normalization steps.
4. Probabilistic Selection: In the final step, a token, denoted as k_out is chosen based on the encrypted probabilities. The selected token can then be fed back into the network for subsequent iterations.

The neural network architectures can vary, with transformers becoming increasingly dominant. In our work, we aim to address the challenges and implications associated with secure probabilistic inference rather than exploring the nuances of efficient neural network architectures, which falls outside the scope of our study. Hence, our primary emphasis is on encrypted probability selection, encompassing the final two steps of converting outputs into encrypted probabilities, as well as making selections based on these encrypted probabilities.

2.2 Fully Homomorphic Encryption (FHE)

Overview. Fully Homomorphic Encryption (FHE) is a class of cryptographic techniques that enables a cloud server to perform computations on encrypted data without the need to decrypt it. FHE schemes usually have four key components: key generation, encryption and decryption algorithms, a set of homomorphic operations, and a bootstrapping operation for lattice-based homomorphic schemes. The key generation process includes the creation of three types of keys: public keys, secret keys, and bootstrapping keys. This key generation step is typically performed by the client and is a one-time computationally intensive setup [33, 9, 28].

During encryption, the public key is used to transform a plaintext message into a many-dimensional ciphertext. In particular, the security of the FHE transformation often relies on hard problems such as LWE [50, 51] and Ring-LWE [39]. Here, some noise is added to the ciphertext to make it cryptographically hard for an attacker to reverse the plaintext-ciphertext mapping. Only a user with a secret key should be able to decrypt the message.

During homomorphic operations, ciphertexts can be manipulated to preserve the meaningfulness of the underlying plaintext. In particular, homomorphic addition and multiplication directly operate on the encrypted data, generating a new output ciphertext; when decrypted using the secret key, the resulting plaintext corresponds to the desired addition or multiplication computation. This property is what makes the encryption scheme "homomorphic." However, each homomorphic operation contributes to noise accumulation in the resulting ciphertexts. If a large number of operations are performed sequentially, the accumulated noise in the ciphertexts may hinder the successful decryption of the data [9].

The bootstrapping operation offers a noise reduction that mitigates this problem. Here, the bootstrapping key can be viewed by the cloud server as the encryption of the secret key. Therefore, the cloud can perform homomorphic decryption and re-encryption, resulting in a new ciphertext with significantly reduced noise. Bootstrapping can be applied repeatedly to facilitate limitless computational depth. However, bootstrapping is computationally expensive and remains the bottleneck for FHE implementations [9, 10].

Homomorphic Cryptosystems and Libraries. BFV [18] and CKKS [7] are two prominent and closely related homomorphic encryption schemes. They are supported by both IBM's HeLIB [23] and Microsoft's SEAL [54] libraries and have the ability to offer integer or fixed-point operations for both real and complex numbers. For neural network implementations, these schemes use polynomial approximations of non-linear functions (e.g., ReLU), which require special retraining. The benefit of these cryptosystems is that they support a compression technique called packing, where multiple values can be encoded in a single ciphertext to increase the throughput of homomorphic operations [3]. However, in this case bootstrapping operations are impractically slow [35], and therefore are not supported in SEAL [32]. Likewise, many BFV and CKKS algorithmic implementations do not use bootstrapping, which is referred to as leveled homomorphic encryption (LHE) [1, 34]. Instead, they adjust the FHE parameters to

handle extra noise, which can reduce latency can increase computation/memory costs.

TFHE is a cryptographic scheme that builds upon the foundations of GSW and its ring variants [8]. It has undergone further advancements and enhancements to become the Concrete Library [57]. TFHE operates on either bits or integers and has an efficient bootstrapping process that can be performed on a scale of milliseconds, which allows for deeper neural networks to be implemented. Our work evaluates the proposed probabilistic selection methods using the TFHE scheme.

2.3 FHE Algorithms

PPML. The fully homomorphic privacy-preserving machine learning (PPML) field has experienced rapid advancement and innovation. In its early stages, initial efforts were confined to shallow neural networks like MNIST. Notably, the SHE technique, introduced in 2019 by Lou and Jiang [38], projected that executing a fully homomorphic AlexNet architecture using TFHE would require over two years. However, in a remarkable four-year span, REDsec [19] managed to accomplish fully homomorphic inference on AlexNet in a mere 1.64 hours.

Despite significant progress in PPML over the recent years, it still remains challenging to implement complex functions homomorphically. First, although the fully connected and convolutional layers have made significant strides in achieving efficient FHE implementations within the integer domain, precision continues to pose limitations. Several libraries, such as REDsec, TAPAS [52], and SHE, have directed their focus toward optimizing efficiency through the use of ternary and binary weights. Additionally, the Concrete-ML library delves into the realm of low-precision weights, ranging from 2 to 8 bits. Employing FHE simulation, this library provides users the flexibility to strike an optimal equilibrium between accuracy and timing while determining the suitable bit width for their specific use case.

Second, the challenge of computing non-linear activation functions over FHE data constitutes another significant hurdle. In the BFV and CKKS schemes, approximations using the Taylor series are frequently employed, or alternative activation functions are adopted, where the simple $f(x) = x^2$ stands out as a prominent choice [20]. In TFHE-based solutions, network architects are able to utilize standard activation functions through a variety of techniques. REDsec introduces bidirectional bridging, a mechanism that transforms integer ciphertexts into encrypted bits; this approach facilitates logic gate implementations of the sign activation function (extracting the negated sign bit) and the ReLU function (performing an AND operation with the negated sign bit and other bits). On the other hand, Concrete introduces an alternative strategy termed programmable bootstrapping (PBS) [10], which empowers users to embed univariate functions within a lookup table. This innovation enables the evaluation of Sigmoid, Tanh, and ReLU functions. In our work, we utilize the more TFHE-friendly "hard sigmoid" approximation, illustrated in Figure 3d. However, this

approach introduces quantization and precision errors that correlate with the programmable bootstrapping lookup table size.

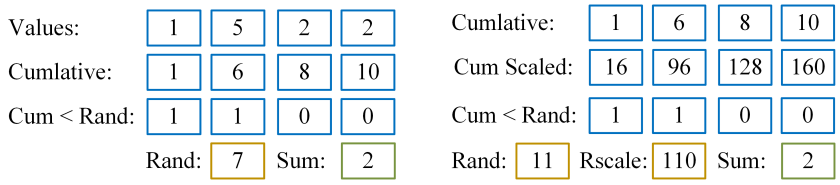
Moreover, the homomorphic computation of the encrypted maximum of two values is possible yet inefficient. This limitation has implications for both MaxPooling operations and the determination of output classifications, where identifying the highest output value becomes challenging. In the context of FHE-PPML, MaxPooling is often circumvented: for instance, Concrete-ML [40], in its version 1.1, lacks MaxPooling support, while REDsec only offers it following a sign activation function. These libraries advocate for the use of SumPooling or AveragePooling instead, albeit at the cost of some accuracy degradation.

Lastly, handling the division of one encrypted number by another encrypted number represents another intricate challenge. In classification tasks, the need for output normalization is not always imperative; the classification outcome can be determined by locating the maximum value. In almost all cases, the responsibility of performing these calculations is delegated to the user post-decryption [19, 38, 40], which effectively renders normalization unnecessary for classification. While this lack of normalization might not considerably impact classification tasks, it does raise significant concerns when grappling with the intricacies of the probabilistic selection problem – a key feature of generative ML and the core area of focus in our investigation. In this context, normalizing outputs becomes crucial, and this paper delves into methodologies designed to tackle this intricate problem, thereby extending the applicability of fully homomorphic encryption into a broader range of machine learning tasks.

ArgMax. The argmax function returns the index of the maximum value in an array, where the closely related argkmax returns the index of the k^{th} largest value in an array. In general, argmax is not considered FHE-friendly since it requires many encrypted comparisons, a costly operation. Most PPML classification works do not perform argmax but instead leave it for the user. Although this approach reduces computation time, it amplifies network bandwidth usage and escalates the vulnerability to model intellectual property theft via black box attacks.

Nevertheless, there is a body of research that covers both argkmax and argmax for TFHE. Notable results in this area include the works by Zuber [59] and Sirdey et. al. [22], later improved by Cong et. al. [12]. These approaches employ the argkmax's sibling, argkmin, for K-nearest neighbor. Overall, they propose two core algorithms: one on that features counting true comparisons, and the second that performs sorting using the batcher sort algorithm.

In our approach, we also employ the batcher-sort network for argmax, also known as the tournament method. Here, pairs of elements are compared and swapped until the max and argmax are found. This has computational complexity $\mathcal{O}(n)$, but can be vectorized to run in $\mathcal{O}(log(n))$. This argmax is the same as the above works, but we implemented ours in the Concrete library instead of TFHE-rust, a lower-level version.



(a) Cumulative Sum Plaintext Probabilistic Selection

(b) Multiplication Cumulative Sum Method Scaling

Fig. 2: **Cumulative Sum**: To select elements with probabilities proportional to their values, we use cumulative sums. In a plaintext implementation (left side), we generate a random number within the range of 0 to the sum of the original vector. In an encrypted domain, where we don't know the sum of the original vector, we employ a multiplication method (right side) with a fixed value, $Rand_{max}$, to generate a range of random numbers spanning the array.

3 Threat Model

Our work addresses the most prevalent scenario in privacy preserving machine learning, where a cloud service provider possesses a model, and users pay to upload their individual inputs and receive generated results from the cloud. An additional layer of complexity arises from the autoregressive generative model paradigm, which requires the cloud to iteratively return model outputs as inputs to the neural network for subsequent iterations.

The central focus of our approach revolves around the protection of user data privacy at the outset, as well as the safeguarding of proprietary network characteristics within the cloud, including model weights and biases. To establish a clear threat model, we operate under the assumption of an honest-but-curious cloud provider that faithfully executes operations on encrypted data but has an incentive to eavesdrop on user data; we also defend against external adversaries to attempt to steal the user data through cyberattacks on the server or the network links.

4 Our Proposed Approaches for Cumulative Sum

In this section we introduce two methods for probabilistic selection based on the cumulative sum operation, while our third method based on the argmax operation is discussed on Section 5. For each proposed algorithm, we address the issues of normalization, precision, and bias.

The cumulative sum method enables us to normalize output values, since there is no efficient method to perform division by an encrypted value in TFHE. The basic steps of the cumulative sum are as follows, using the example of Fig. 2a:

1. **Activation:** This step assumes all values are non-negative, although not necessarily normalized. Typically this is done with the softmax function in neural networks, which is a normalized sigmoid function. In the encrypted domain, a hard sigmoid can be used as an approximation to sigmoid. In

our approach, we use the hard sigmoid activation function to transform the values into non-negative.
2. **Cumulative Sum:** This step calculates the cumulative sum of the post-activation neural network output values. While this operation is sequential, addition is very efficient in FHE.
3. **Random Number Generation:** The plaintext version of this algorithm would generate a random number between 0 and the sum of neural network output values, which can be taken from the previous step. However, using FHE this sum is encrypted, which brings a new challenge on how we can approximate the random number generation.
4. **Comparison:** In this step, we run an encrypted comparison of the random number with each index in the cumulative sum array. If the cumulative sum is less than the random number, we return 1, otherwise 0. This operation is vectorized.
5. **Summation:** Finally, we conclude the process by summing the comparison output, yielding the (encrypted) index of the token k_i, which can subsequently be reintroduced into the neural network. Notably, for networks that require one-hot vector inputs, we can subtract the comparison output shifted by one position from the original comparison output, deriving the desired one-hot vector.

In theory, this methodology would provide an unbiased way for encrypted probabilistic selection. However, three main issues prevent this algorithm's success in the encrypted domain. First, some form of **normalization** is needed. In the plaintext algorithm, this manifests in selecting a random number between 0 and the sum of these values, which is already calculated as the last index of the cumulative sum. However, when the value of the sum is encrypted, generating a random number from 0 to this encrypted sum is no longer possible. Therefore, our two proposed cumulative sum methods differ on the way this is achieved. Second, a **bias** may be introduced from approximation, rounding and precision errors, and for each method, we evaluate this bias. Lastly, FHE computation suffers from **precision** problems, requiring users to choose between accuracy and computational complexity. Therefore, careful tuning is required to obtain the optimal balance of these two constraints.

4.1 Multiplication-Scaled Cumulative Sum

Methodology. The main challenge to tackle for the cumulative sum method is generating a random number between 0 and the maximum value of the encrypted cumulative sum array, which we denote as $R_{ideal} = rand(0, Cumsum_{max})$. In the multiplication-scaled method, we first select a fixed range for the random number, where $R_{fixed} = rand(0, Rand_{max})$. In the encrypted domain, we can then enforce the scale to be the product of the maximum possible random value and the maximum value of the array, $Cumsum_{max} \cdot Rand_{max}$. This is done by multiplying each value of the cumsum array by $Rand_{max}$, and by multiplying our random number by $R_{scaled} = R_{fixed} \cdot Cumsum_{max}$.

Normalization. This method achieves normalization by adjusting the scale as described above. This normalizes the output and random number to $Cumsum_{max} \cdot Rand_{max}$.

Bias. In a plaintext version of this algorithm with floating point precision or significantly large values of $Rand_{max}$, there is little bias since the cumulative sum operates a normalization alternative. However, in the encrypted domain with a small, low precision value of $Rand_{max}$, the heavy discretization of the random variables introduces a bias.

Figure 2b illustrates an example of such bias. The possible values for R_{scaled} are {0, 10, 20, 30...150}. This leads to probabilities of $\{\frac{2}{16}, \frac{8}{16}, \frac{3}{16}, \frac{3}{16}\}$, which are slightly skewed from the original $\{\frac{1}{10}, \frac{5}{10}, \frac{2}{10}, \frac{2}{10}\}$. Such patterns in the data will cause consistent skew toward certain indexes, and will always increase the likelihood of the first index. To prevent these patterns in the data, we propose shuffling the order of the array before applying the cumulative sum. In FHE, we can recover the original index with a simple private information retrieval (PIR) lookup in $\mathcal{O}(n)$ time.

Finally, there is a natural smoothing effect that comes from a combination of rounding and the hard sigmoid approximation set to 0. This creates a long tail effect that does not scale for large datasets, as each 0 value has a smoothing value of $1/(N \cdot Rand_{max})$.

Precision and Complexity. This method has $\mathcal{O}(n)$ multiplications, $\mathcal{O}(n)$ additions and $\mathcal{O}(n)$ comparisons, where n is the vocabulary size (i.e., the number of possible tokens). However, the cumulative sum operation requires high precision to begin with, and multiplying by $Rand_{max}$ only increases this need. Furthermore, $Rand_{max}$ cannot be a static value. To prevent bias, $Rand_{max}$ should increase with cumulative sum. The precision, representing the number of bits required to represent the largest number, is $\mathcal{O}(\log(Rand_{max}) + \log(n))$ bits, so the evaluation times using FHE are less scalable to larger datasets.

Experimental Characterization. In Figure 3a, we present two examples of probability distributions for the encrypted cumulative sum multiplication-scaled method vs. its plaintext counterparts. For the plaintext case, we sorted the output tokens by their selection frequency, and compare with the cumulative sum equivalent distribution for the same ordering. In the average case, we observe that our model could not achieve a very high selection probabilities; specifically, we observe a shelf (around 0.16) indicating discretization (i.e., ML outputs rounded to the same value). For the worst case, all neural network outputs are discretized to 0, creating a uniform distribution instead of the desired distribution.

Both of these test cases could potentially be enhanced as the proficiency of the Concrete library toward handling higher precision inputs improves. Nevertheless, it's crucial to note that this approach exhibits a substantial appetite for precision, making it less favorable in comparison to other algorithms that demand fewer precision-intensive resources.

4.2 Heuristic Cumulative-Sum

Methodology. This method avoids scaling the random multiplier by the sum of post-activation ML outputs, but instead approximates $Rand_{max}$ to be based on data set heuristics. We assume that the dataset is randomly shuffled, as described in the multiplication-scaled cumulative sum method.

Normalization. Our pseudo-norm is based on heuristics and is not exactly normalized. If the $Rand_{max}$ constant is too small in this method, it will only select the few values at the front of the shuffled array. If the $Rand_{max}$ constant is too large, then the system is biased heavily toward the last single element in the shuffled array. Since selecting a $Rand_{max}$ constant that is too small spreads the bias out on multiple values instead of a single value, it is best to choose a $Rand_{max}$ constant closer to the minimum. In our methodology, the $Rand_{max}$ constant is the cumulative sum value in our dataset's 10^{th} percentile.

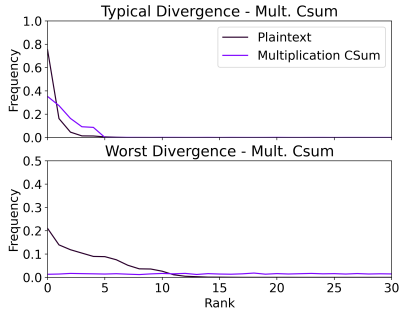
Bias. The introduced bias depends on the standard deviation of the sum of post-activation ML outputs in the dataset. With higher standard deviations, the $Rand_{max}$ constant will deviate further from the cumulative sum of the network output, causing larger biases. Unfortunately, this was the typical case for our target neural networks, so that, combined with the low precision, it set some of the test set cases to all 0s. Moreover, this approach still incurs smoothing and long tail biases for small values of $Rand_{max}$, which causes increased bias for larger dataset sizes.

Precision and Complexity. This algorithm uses $\mathcal{O}(n)$ additions and $\mathcal{O}(n)$ comparisons. Here, the maximum precision required is reduced due to the lack of multiplication-scaling but is still dependent on the sum of neural network outputs, which is $\mathcal{O}(\log(n))$ bits.

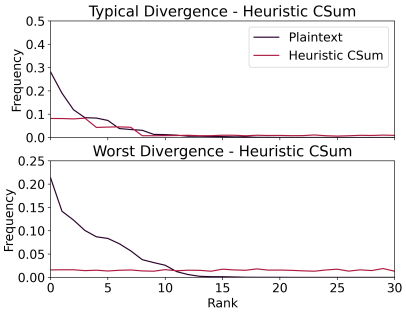
Experimental Characterization As shown in Figure 3b, this method still encounters many of the same issues reported for the multiplication-scaled cumulative sum method. Moreover, precision is impacted as the vocabulary size grew; however, there is still room to increase precision at the cost of slowing down execution time. Our analysis shows that the output of the ML model had high variability, which caused outputs where the entire vector was discretized to 0s.

5 Our Proposed Approach for Argmax

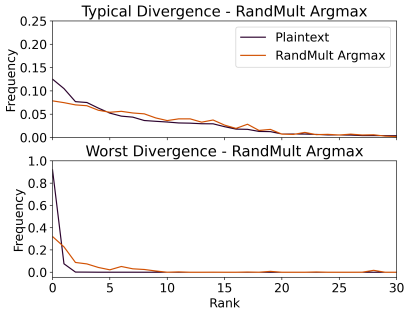
While argmax can be useful for ML tasks like classification, its deterministic nature is not very useful for generative AI. Nevertheless, if some randomness can be added to the output, it is possible to achieve the desired stochastic result. This is the core idea behind our proposed argmax method, as we also aim to lower the precision required. Towards that end, our methodology leverages the argmax tournament method, which can be interpreted as a derivative of batcher sort.



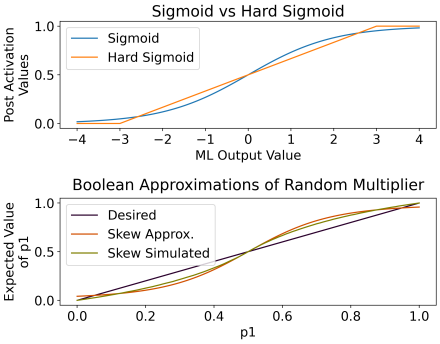
(a) Multiplication Cumulative Sum Divergence



(b) Heuristic Cumulative Sum Divergence



(c) Random Multiplier Argmax Divergence



(d) Approximations: Hard Sofmax and RandMult Method

Fig. 3: **Divergence from Plaintext:** This figure highlights the constraints imposed by our approximation techniques and precision errors. In the case of cumulative sums, precision accumulates, and the enforcement of low precision results in only several discretized output values. This can also lead to scenarios where the output vectors a_i are rounded to all zeros, which the algorithm interprets as uniform random selection. On the other hand, the RandMult method exhibits a greater capacity to accommodate higher precision levels and circumvents the biases stemming from random number generation, which are problematic in the cumulative sum approach. However, it's essential to acknowledge that all these methods encounter challenges in achieving peak accuracies due to rounding errors in the activation function.

5.1 Random Multiplication Argmax Method

Methodology. In the random multiplication method, multiplying the post-activation ML outputs with a random vector can be used as the input to the argmax function. This will generate a different answer everytime, although it produces some skew for probabilistic selection for the plaintext variant.

Normalization. The normalization constraint is mitigated when using argmax methodology. Specifically, we no longer not need to normalize the outputs if we are only seeking the maximum.

Bias. Unlike the cumulative sum generator, this approach introduces a distortion that sharpens the probability distribution. More concretely, suppose we multiply encrypted output distributions $\{p_1, p_2\}$ by uniform random variables $\{\hat{X_1}, \hat{X_2}\}$, respectively. It is important to note that in this context, the variables p_1 and p_2 represent "probabilities" but do not necessarily need to be normalized to one. We wish to find the probability that $p_1 \cdot \hat{X_1}$ is larger than $p_2 \cdot \hat{X_2}$, represented by

$$\mathcal{P}(D_{1,2} > 0) = \mathcal{P}(p_1 \cdot \hat{X_1} - p_2 \cdot \hat{X_2} > 0), \tag{1}$$

where $D_{1,2} = (p_1 \cdot \hat{X_1} - p_2 \cdot \hat{X_2})$. To calculate this probability, we can first find the expected value

$$\mathbb{E}(D_{1,2}) = \int_0^1 (p_1 \cdot \hat{X_1}) d\hat{X_1} - \int_0^1 (p_2 \cdot \hat{X_2}) d\hat{X_2} = \frac{p_1 - p_2}{2} \tag{2}$$

and variance

$$\mathbb{V}(D_{1,2}) = \int_0^1 \int_0^1 (p_1 \cdot \hat{X_1} - p_2 \cdot \hat{X_2})^2 d\hat{X_1} d\hat{X_2} = \frac{2 \cdot p_1^2 + 2 \cdot p_2^2 - 3 \cdot p_1 \cdot p_2}{6}. \tag{3}$$

With the mean and variance, we can use the normal cumulative distribution function (cdf) $\mathcal{N}\left(\frac{0-\mathbb{E}(D_{1,2})}{\mathbb{V}(D_{1,2})}\right)$ to find the probabilities for different values of p_1.

This gets harder to extrapolate with more variables since the probabilities are not independent, and several constraints between pairs of variables need to be met (i.e., $p_1 > p_2$, $p_2 > p_3$, $p_3 > p_1$ is a contradiction). During algorithmic development, we utilize the bayesian form where contradicted states are removed (the denominator does not sum to 1 when $N > 2$). Thus, the probability token k_1 is selected given probability p_1 is:

$$\frac{\Pi_{i=1}^{N} \mathcal{N}\left(\frac{0-\mathbb{E}(D_{1,i})}{\mathbb{V}(D_{1,i})}\right)}{\sum_{j=1}^{N} \Pi_{i=1, i\neq j}^{N} \mathcal{N}\left(\frac{0-\mathbb{E}(D_{j,i})}{\mathbb{V}(D_{j,i})}\right)}. \tag{4}$$

To illustrate this skew, Figure 3d shows the boolean case, defined as p_1 and $p_2 = 1 - p_1$. The figure shows both the ideal and the distorted probabilities. A slight distortion makes these variables sharper; however, this works in our favor as it reintroduces the "S" curve lost through FHE-friendly hard softmax approximation. *This is a surprising result that helps our RandMax algorithm to cancel out the bias introduced through the activation function approximation and achieve a result close to plaintext evaluation.*

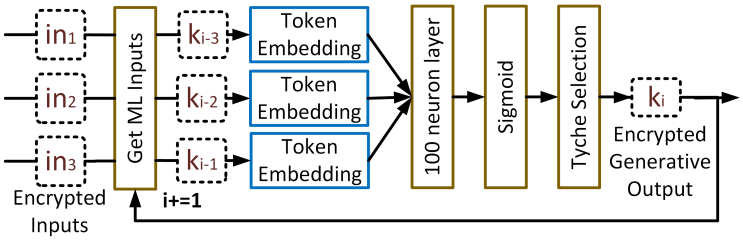


Fig. 4: **Our neural network architechture:** We use a simple mult-layer perception (MLP) model with an embedding table of size 10 and a hidden layer consisting of 100 hidden neurons. We use 3-ngram characters as inputs and the output is the number of possible tokens (i.e., vocabulary size). This output is fed through our probabilistic selection methodology to select a generated encrypted output.

Precision and Complexity. As mentioned in Section 2.3, the computational complexity of argmax involves $\mathcal{O}(n)$ comparisons. The multiplication step adds $\mathcal{O}(n)$ multiplications. The highest value is the maximum value of the neural network output times $Rand_{max}$ bits. Therefore, the precision is $\mathcal{O}(log(Rand_{max}) + log(max(v))$ bits, as no values are accumulated in the argmax computation. This is much more scalable than the cumulative sum methods, since $max(v) << sum(v)$, and this bitsize does not grow with the vocabulary size.

Experimental Characterization. This method performs significantly better on larger datasets, which is attributed to the lack of precision bitsize growth; this allows our methodology to achieve a higher precision overall. Even with this method's sharpening bias, the model may still not be able to meet the steep probabilities expected from the outputs, as seen in Figure 3c. However, the argmax curve approximates the expected distribution more accurately than cumulative sum methods and achieves a more accurate result with FHE.

6 Experimental Evaluation

6.1 Description of our Datasets

Our generative language models focused on generating letter tokens. This enables us to implement a small encrypted multilayer perceptron (MLP) network, illustrated in Figure 4. This network architecture was kept consistent across all of our datasets and tests to ensure fair comparison of our results. The network was run encrypted using TFHE, and the results were utilized for our probabilistic selection experiments. Our experiments comprise five different datasets, as characterized below:

SSA Names Dataset [30]: This dataset comprises 32K of the most common names taken from ssa.gov for the year 2018. It contains very short phrases (a

single name), and has much less training data and higher entropy than our other datasets. This leads to a higher loss score. There are 27 tokens in this dataset, one for each letter of the English alphabet (all lowercase) and one stop character.

Shakespeare Dataset [29] (Lowercase): This dataset consists of all of the works of Shakespeare. We modify this dataset to turn all letters to lowercase. In total, there are 39 tokens in this dataset including the 26 letters of the English alphabet and miscellaneous punctuation.

Shakespeare Dataset [29] (All Case): This dataset is the same as above but regular uppercase and lowercase letters are used for 65 tokens in total. Uppercase letters have a higher level of predictability, leading to lower entropy in this dataset than its lowercase counterpart.

German Parliament (Lowercase): We created this dataset to test the scalability of our algorithms. It consists of proceedings of the German parliament, in the German language. Numbers, umlauts, and characters used in formal government writing such as "§" and parenthesis expand this lowercase dataset to 74 characters. The German language is more predictable than English for length 3 n-grams [54], which leads to more predictable results and less of a long-tail effect than the other datasets.

German Parliament (All Case): This dataset is the same as above, but with 29 extra uppercase characters, bringing the dataset size to 103 tokens.

6.2 Concrete Library for TFHE

For our experiments, we utilized the Concrete library, which implements the TFHE scheme. While Concrete offers a mature interface and compiler, the current version has two major limitations related to parallelism. First, the library only supports vector-level CPU parallelization using a curated list of numpy primitives. Outside of these numpy library functions, there is little parallelization support, and calculations are limited to one CPU core. This includes for-loops and homomorphic operations not being parallelized, even though there is plenty of opportunity to do so and has been done in prior work [13, 43, 19].

Second, the Concrete compiler does not scale well when using for-loops, which must be unrolled and evaluated sequentially. Due to this limitation, our Concrete implementation of batchersort was unable to support larger vectors. Consequently, our results are focused on argmax instead of argkmax, as evaluation of argkmax was not feasible.

6.3 Hardware Platform for Evaluations

For our experimental evaluation we use an r5.24xlarge server on AWS. To ensure our results are comparable across datasets, we calculate the amortized cost across 10 iterations run in parallel. This allows small vocabulary-size datasets to utilize all 96 cores during vector operations, ensuring a fair comparison, but only 10 cores were used in the sequential parts of our algorithms.

Table 1: **Timing Results (seconds)**: In Table 1b, we report how our three proposed techniques scale with time. The Cumulative Sum methods scale more rapidly, and seem very dependent on the worst-case CumSum magnitude (labeled Max CSum). The RandMult method, albeit longer, scales much more linearly and predictably. In Table 1a, we provide additional context on state-of-the-art TFHE PPML latencies; these could be the ML architectures that could feed into our proposed methodologies.

(a) TFHE PPML

Inference	Time
MNIST [19]	1 min.
VGG-11 [19]	40 min.
ImageNet [19]	2 hrs.
Attention[a] [41]	3 min.
GPT3 [24]	58 days[b]
Our MLP	8 min.

(b) Timing Results for our Methodologies

			Timing Results		
Dataset	Vocab Size	Max CSum	Mult. CSum	Heuristic CSum	RandMult Argmax
Names	27	41	102	4.0	156
Low. Shakes.	39	84	265	21.8	224
Shakespeare	65	78	379	107	348
Low. German	74	19	228	119	386
German	103	10	294	151	400

[a]Single attention head. For comparison, ChatGPT-3 has 96 attention heads for each of its 96 layers.
[b]Extrapolated time based on 10^9 bootstraps, 200 bootstraps per second.

6.4 Latency Performance

Runtime performance using FHE depends on algorithm complexity and required precision. In this case, three different factors need to be taken into consideration for our experiments.

Precision and Dataset Characteristics: The output characteristics of the dataset play a key role in runtime performance. In particular, Concrete tunes the TFHE parameters based on the worst-case precision in the plaintext test cases, and uses these parameters for both evaluation and encryption. For the German dataset specifically, the sum of the outputs was significantly lower, dropping the required precision for the cumulative sum operation from 7-bits to 5-bits. This caused a dramatic decrease in runtime for the multiplication cumulative sum method, whose biggest limitation was precision constraints. The cumsum heuristic also benefited from this characteristic, causing a decrease in latency.

Computational Complexity: As the vocabulary size grows, the cumulative sum methods are expected to grow faster than the argmax method due to the inherent computational and precision-related complexity. However, the cumulative sum methods are more influenced by the output characteristics of the dataset, particularly the largest sum of the post-activation ML output vector across test cases. Randmax growth is much more predictably since it does not rely on this dependency. Still, looking at how size influences runtime performance, the cumulative sum methods start out really efficient and grow to increased runtime overheads. Therefore, the randmax algorithm is a better fit as

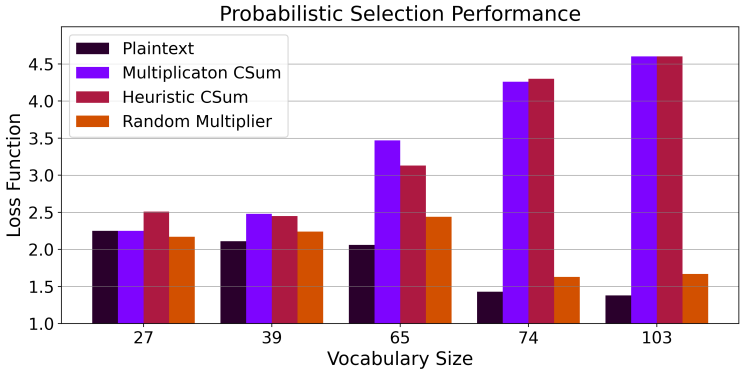


Fig. 5: **Loss:** We compare how our three proposed methods compare with respect to accuracy. Due to the long tail caused by low precision rounding errors, the cumulative sum has an increasing loss as the vocabulary size grows. Conversely, the random multiplier preserves the low model loss as the vocabulary size grows.

dataset sizes grow, and for higher entropy datasets that have many high-valued outputs.

Parallelism: The cumulative sum methods have the best parallelism; only FHE-friendly additions are sequential. Conversely, the random multiplication argmax approach requires comparisons to be run sequentially or using the tournament method. With the current implementation of the Concrete library, which only performs well on vectorized operations, the random multiplication method has the lowest CPU resource utilization among our results. Improvements to the library, such as a built-in argmax function and homomorphic level parallelism, would further reduce the timing overhead of our methods.

6.5 Model Performance

The cumulative sum methods showed increasing loss with the dataset size, due to the smoothing effect creating a long tail. Thus, as the dataset grows, more precision is needed to distinguish between probable and improbable values. There was also a higher occurrence of all zero values with the German datasets, resulting from the dataset characteristics and vector size.

The random multiplier argmax method does not suffer from this long tail limitation and can offer higher precision that does not grow with the vector size, resulting in better performance from larger datasets. In addition, the random multiplier dataset is able to support higher input precision, since the max precision is capped to $Rand_{max} \cdot \max(\text{hard sigmoid}(x) = 1$, unlike the cumulative sum methods that can grow and have a theoretical upper bound equal to the vocabulary size n.

7 Discussion of Related Works

To the best of our knowledge, this is the first work to perform probabilistic selection for language models. To give additional context to our approach, we look at three categories of related works. The first is encrypted language models, which gives some context into the orthogonal work and latencies of the upstream ML algorithms. The second category of related work is enhanced LLMs, which primarily use forms of obfuscation to hide query data. The final category we compare with are TFHE decision trees, which can be stochastic or even generative in nature.

7.1 Encrypted Language Models

Zama, the designers of the Concrete library, have developed two text-based models. The first one involves sentiment analysis classification [42]. Since this is a classification problem, they do not invoke probabilistic selection, unlike our work, and they use a simple XGBoost to classify the data. They attempt two methodologies for *unencrypted* text preprocessing, one based on term frequency-inverse document frequency (tfidf) and a second using the RoBERTa transformer excluding the final layer.

Zama has also developed a transformer model [41]. Their first implementation is a single transformer block with a single-head GPT2 variant, where layers 2 through 11, over 90% of their model, are run in plaintext. Their second implementation is a multi-head variant with 12 attention heads. This is still implemented as a single layer and with lower precision. The embedding table, layer normalization and probabilistic selection of the words is also performed in plaintext. This is in contrast to our techniques, which allow running every part of the generative AI encrypted.

THE-X is another work that attempts to build a homomorphic transformer, but they remove much of the homomorphic work from the server, including activations, and instead ask the user perform these in the plaintext domain. This defeats the purpose of using HE, as MPC would be a better choice if the client needs to perform serious computations [6].

For MPC language models, which we discuss here for completeness, many works were recently published that seek to optimize the inference of the BERT-based language models. Iron [23] was the first work on MPC transformers, able to achieve inference speeds on RoBERTa in one minute of online computation. Primer [58] is another work that merges ciphertext operations and achieves a latency of around 40 seconds. East [17] and Liu [37] both replace standard functions with MPC-friendly counterparts and achieve similar performance to Primer. These MPC works all focus on building fast transformers, but none of them addresses the issue of probabilistic selection. Furthermore, MPC contains at least two computing parties, compared to the FHE case of a single idle client outsourcing computation to a single computing server. Therefore, MPC must either invoke the user or have two non-colluding parties perform the computation, which is a different threat model and use case compared to FHE.

7.2 Privacy of Cloud Language Models

There are several techniques that can be used to assist model privacy. The first is differential privacy, which consists of obfuscating the inputs of an input query. For word tokens, this involves processing the text by replacing synonyms and redacting any sensitive information [5, 53, 36]. However, this technique can harm accuracy, and while the users' direct text may be altered, an attacker can still distill the meaning behind the original query [4].

A second approach entails projecting the inputs into a related subspace. A common technique includes sending a compressed token embedding instead of raw words, which is a lower-dimensional version of the inputs [45]. This work also recommends obfuscation by the rounding of plaintext floating point numbers (which TFHE is very good at in the encrypted domain). These techniques may make the input unreadable to a human attacker, but the meaning of these inputs can still be extracted.

7.3 TFHE Decision Trees

The basic idea behind decision trees is that decisions can be made at each node until a leaf is reached, and these decisions can be made probabilistically. In the TFHE scenario, the model owner has a set of weights which they encrypt and a user sends a set of probabilities. Then, for each branch, the two probabilities are compared, and the final value can be calculated as $\sum_{i=0}^{N} \Pi_{j:b_j \in \text{Path}(v_i)} b_j \cdot v_i$, where b is a true-false branch decision and v is the leaf value. While these works do not focus on generative ML, they employ some form of probabilistic selection that is worth discussing.

Many recent works have promoted decision trees. These include Paul et. al. [46], which uses TFHE's programmable bootstrapping feature to make decisions; SortingHat [11], which focuses on transciphering of decision tree inputs; Probonite [2] which merges decision tree branches in a process they call "blinding". In addition, Concrete ML [40] has TFHE-encrypted decision tree and random forest APIs available.

None of the models above, however, have been adapted for generative models. They would need to address the challenge of appending the data to the tree and normalize the probabilities at each branch. This issue has yet to be addressed for TFHE decision trees, but was a core contribution of our work. This additional computation would incur high runtime overheads, as well as require higher amounts of precision.

8 Future Work

This work utilizes the new Concrete THFE-based library, which is still rapidly evolving. There are several Concrete features that are still under development that would be of great interest to this work.

The first is an efficient implementation of the maximum, argmax and argkmax functions. In our methodology, we were not able to implement state-of-the-art argkmax techniques due to inefficiencies in the Concrete compiler that led to days-long compile times. These techniques have been proven in THFE-rust, but have not migrated to Concrete. The authors of the library confirmed there are no plans to allow executing code written in TFHE-rust from Concrete, but they plan to implement the argmin and argmax as part of K-nearest neighbor.[1] A better Concrete implementation would speed up our argmax random multiplier implementation, while having argkmax available would allow us to explore additional algorithms in this space.

Second, each TFHE operation, such as a single addition, is a complex lattice-based computation that can be parallelized. There are many TFHE research works that accelerate homomorphic operations, such as cuFHE[13], nuFHE [43], REDcuFHE [19], and ArctyrEX [21] that parallelize the lower level operations of TFHE on GPUs, but Concrete does not yet offer this functionality. There are also several works of TFHE hardware accelerators, with MATCHA [27] being the most recent work in the FPGA/ASIC space. We look forward to obtain a greater speedup and utilize more CPU bandwidth through this greater parallelization effort.

9 Concluding Remarks

In this work, we introduce a novel methodology for encrypted probabilistic selection in TFHE. Our approach, we compare two possible methods, the cumulative sum and the argmax, and provide three algorithms for probabilistic selection optimized for generative language models over encrypted data. Our findings show that the argmax-based random multiplication method outperforms the cumulative sum methods in terms of loss stability and precision required, despite offering some bias in the plaintext domain. This result opens new applications into private generative ML, and complements much of the existing work on adapting ML algorithms for TFHE. With many new TFHE developments on the horizon, this work is among the first address one of the main challenges toward encrypted generative AI.

References

1. Aharoni, E., Adir, A., Baruch, M., Drucker, N., Ezov, G., Farkash, A., Greenberg, L., Masalha, R., Moshkowich, G., Murik, D., et al.: Helayers: A tile tensors framework for large neural networks on encrypted data. arXiv preprint arXiv:2011.01805 (2020)
2. Azogagh, S., Delfour, V., Gambs, S., Killijian, M.O.: Probonite: Private one-branch-only non-interactive decision tree evaluation. In: Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography. pp. 23–33 (2022)

[1] Citation to public forum discussion omitted for anonymity of the authors of this paper.

3. Benaissa, A., Retiat, B., Cebere, B., Belfedhal, A.E.: Tenseal: A library for encrypted tensor operations using homomorphic encryption. arXiv preprint arXiv:2104.03152 (2021)
4. Brown, H., Lee, K., Mireshghallah, F., Shokri, R., Tramèr, F.: What does it mean for a language model to preserve privacy? In: Proceedings of the 2022 ACM Conference on Fairness, Accountability, and Transparency. pp. 2280–2292 (2022)
5. Carranza, A.G., Farahani, R., Ponomareva, N., Kurakin, A., Jagielski, M., Nasr, M.: Privacy-preserving recommender systems with synthetic query generation using differentially private large language models. arXiv preprint arXiv:2305.05973 (2023)
6. Chen, T., Bao, H., Huang, S., Dong, L., Jiao, B., Jiang, D., Zhou, H., Li, J., Wei, F.: The-x: Privacy-preserving transformer inference with homomorphic encryption. arXiv preprint arXiv:2206.00216 (2022)
7. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23. pp. 409–437. Springer (2017)
8. Chillotti, I., Gama, N., Georgieva, M., Izabachène, M.: Tfhe: fast fully homomorphic encryption over the torus. Journal of Cryptology **33**(1), 34–91 (2020)
9. Chillotti, I., Joye, M., Ligier, D., Orfila, J.B., Tap, S.: Concrete: Concrete operates on ciphertexts rapidly by extending tfhe. In: WAHC 2020-8th Workshop on Encrypted Computing & Applied Homomorphic Cryptography (2020)
10. Chillotti, I., Joye, M., Paillier, P.: Programmable bootstrapping enables efficient homomorphic inference of deep neural networks. In: Cyber Security Cryptography and Machine Learning: 5th International Symposium, CSCML 2021, Be'er Sheva, Israel, July 8–9, 2021, Proceedings 5. pp. 1–19. Springer (2021)
11. Cong, K., Das, D., Park, J., Pereira, H.V.: Sortinghat: Efficient private decision tree evaluation via homomorphic encryption and transciphering. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 563–577 (2022)
12. Cong, K., Geelen, R., Kang, J., Park, J.: Efficient and secure k-nn classification from improved data-oblivious programs and homomorphic encryption. Cryptology ePrint Archive (2023)
13. Dai, W., Sunar, B.: cuFHE (v1.0). https://github.com/vernamlab/cuFHE (2018)
14. De, S., Bermudez-Edo, M., Xu, H., Cai, Z.: Deep generative models in the industrial internet of things: a survey. IEEE Transactions on Industrial Informatics **18**(9), 5728–5737 (2022)
15. Derose, A.: These companies have banned or limited ChatGPT at work. Morning Brew (May 2023)
16. Devlin, J., Chang, M.W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)
17. Ding, Y., Guo, H., Guan, Y., Liu, W., Huo, J., Guan, Z., Zhang, X.: East: Efficient and accurate secure transformer framework for inference. arXiv preprint arXiv:2308.09923 (2023)
18. Fan, J., Vercauteren, F.: Somewhat practical fully homomorphic encryption. Cryptology ePrint Archive (2012)
19. Folkerts, L., Gouert, C., Tsoutsos, N.G.: REDsec: Running Encrypted Discretized Neural Networks in Seconds. In: Network and Distributed System Security Symposium (NDSS). pp. 1–17 (2023)

20. Gilad-Bachrach, R., Dowlin, N., Laine, K., Lauter, K., Naehrig, M., Wernsing, J.: Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy. In: International Conference on Machine Learning. pp. 201–210. PMLR (2016)
21. Gouert, C., Joseph, V., Dalton, S., Augonnet, C., Garland, M., Tsoutsos, N.G.: Arctyrex: Accelerated encrypted execution of general-purpose applications. arXiv preprint arXiv:2306.11006 (2023)
22. Grivet Sébert, A., Pinot, R., Zuber, M., Gouy-Pailler, C., Sirdey, R.: Speed: secure, private, and efficient deep learning. Machine Learning **110**, 675–694 (2021)
23. Hao, M., Li, H., Chen, H., Xing, P., Xu, G., Zhang, T.: Iron: Private inference on transformers. Advances in Neural Information Processing Systems **35**, 15718–15731 (2022)
24. Hindi, R.: Making chatgpt encrypted end-to-end (Apr 2023), https://www.zama.ai/post/chatgpt-privacy-with-homomorphic-encryption
25. Huallpa, J.J., et al.: Exploring the ethical considerations of using chat gpt in university education. Periodicals of Engineering and Natural Sciences **11**(4), 105–115 (2023)
26. JaxonAI: Companies that have banned ChatGPT (Jun 2023), https://jaxon.ai/list-of-companies-that-have-banned-chatgpt/
27. Jiang, L., Lou, Q., Joshi, N.: Matcha: A fast and energy-efficient accelerator for fully homomorphic encryption over the torus. In: Proceedings of the 59th ACM/IEEE Design Automation Conference. pp. 235–240 (2022)
28. Joye, M.: Tfhe public-key encryption revisited. Cryptology ePrint Archive (2023)
29. Karpathy, A.: char-rnn. https://github.com/karpathy/char-rnn (2015)
30. Karpathy, A.: Makemore Dataset and Network (2022), https://github.com/karpathy/makemore
31. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems **25**, 1097–1105 (2012)
32. Laine, K.: Simple encrypted arithmetic library 2.3. 1. Microsoft Research https://www. microsoft. com/en-us/research/uploads/prod/2017/11/sealmanual-2-3-1. pdf (2017)
33. Lee, C., Min, S., Seo, J., Song, Y.: Faster tfhe bootstrapping with block binary keys. Cryptology ePrint Archive (2023)
34. Lee, J.W., Kang, H., Lee, Y., Choi, W., Eom, J., Deryabin, M., Lee, E., Lee, J., Yoo, D., Kim, Y.S., et al.: Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. IEEE Access **10**, 30039–30054 (2022)
35. Lee, J.W., Lee, E., Lee, Y., Kim, Y.S., No, J.S.: High-precision bootstrapping of rns-ckks homomorphic encryption using optimal minimax polynomial approximation and inverse sine function. 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 40. pp. 618–647. Springer (2021)
36. Li, Y., Tan, Z., Liu, Y.: Privacy-preserving prompt tuning for large language model services. arXiv preprint arXiv:2305.06212 (2023)
37. Liu, X., Liu, Z.: Llms can understand encrypted prompt: Towards privacy-computing friendly transformers. arXiv preprint arXiv:2305.18396 (2023)
38. Lou, Q., Jiang, L.: SHE: A Fast and Accurate Deep Neural Network for Encrypted Data. Advances in Neural Information Processing Systems **32**, 10035–10043 (2019)

39. Lyubashevsky, V., Peikert, C., Regev, O.: On ideal lattices and learning with errors over rings. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 1–23. Springer (2010)
40. Meyre, A., Chevallier-Mames, B., Frery, J., Stoian, A., Bredehoft, R., Montero, L., Kherfallah, C.: Concrete ML: a privacy-preserving machine learning library using fully homomorphic encryption for data scientists (2022), https://github.com/zama-ai/concrete-ml
41. Meyre, A., Chevallier-Mames, B., Frery, J., Stoian, A., Bredehoft, R., Montero, L., Kherfallah, C.: Secure large language models using fully homomorphic encryption (fhe). https://github.com/zama-ai/concrete-ml/blob/release/1.1.x/use_case_examples/llm (2023)
42. Meyre, A., Chevallier-Mames, B., Frery, J., Stoian, A., Bredehoft, R., Montero, L., Kherfallah, C.: Sentiment analysis with fhe. https //github.com/zama-ai/concrete-ml/blob/release/1.1.x/use_case_examples/ sentiment_analysis_with_transformer/SentimentClassification.ipynb (2023)
43. NuCypher: nuFHE (v0.0.3). https://github.com/nucypher/nufhe (2019)
44. Van den Oord, A., Kalchbrenner, N., Espeholt, L., Vinyals, O., Graves, A., et al.: Conditional image generation with pixelcnn decoders. Advances in neural information processing systems **29** (2016)
45. Pan, X., Zhang, M., Ji, S., Yang, M.: Privacy risks of general-purpose language models. In: 2020 IEEE Symposium on Security and Privacy (SP). pp. 1314–1331. IEEE (2020)
46. Paul, J., Tan, B.H.M., Veeravalli, B., Aung, K.M.M.: Non-interactive decision trees and applications with multi-bit tfhe. Algorithms **15**(9), 333 (2022)
47. Pearce, H., Ahmad, B., Tan, B., Dolan-Gavitt, B., Karri, R.: Asleep at the keyboard? assessing the security of github copilot's code contributions. In: 2022 IEEE Symposium on Security and Privacy (SP). pp. 754–768. IEEE (2022)
48. Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al.: Improving language understanding by generative pre-training (2018)
49. Ray, P.P.: Chatgpt: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope. Internet of Things and Cyber-Physical Systems (2023)
50. Regev, O.: On lattices, learning with errors, random linear codes, and cryptography. Journal of the ACM (JACM) **56**(6), 1–40 (2009)
51. Regev, O.: The learning with errors problem. Invited survey in CCC **7**(30), 11 (2010)
52. Sanyal, A., Kusner, M., Gascon, A., Kanade, V.: TAPAS: Tricks to accelerate (encrypted) prediction as a service. In: International Conference on Machine Learning. pp. 4490–4499. PMLR (2018)
53. Shi, W., Cui, A., Li, E., Jia, R., Yu, Z.: Selective differential privacy for language modeling. arXiv preprint arXiv:2108.12944 (2021)
54. Smith, R.: Distinct word length frequencies: distributions and symbol entropies. Glottometrics 23 p. 7 (2012)
55. Taylor, T.: The top types of ai-generated content in marketing (Oct 2023), https://blog.hubspot.com/marketing/top-types-of-ai-generated-content-in-marketing
56. Telford, T., Verma, P.: Employees want ChatGPT at work. Bosses worry they'll spill secrets. The Washington Post (Jul 2023), https://www.washingtonpost.com/business/2023/07/10/chatgpt-safe-company-work-ban-lawyers-code/

57. Zama: Concrete: TFHE Compiler that converts python programs into FHE equivalent (2022), https://github.com/zama-ai/concrete
58. Zheng, M., Lou, Q., Jiang, L.: Primer: Fast private transformer inference on encrypted data. arXiv preprint arXiv:2303.13679 (2023)
59. Zuber, M., Sirdey, R.: Efficient homomorphic evaluation of k-nn classifiers. Proc. Priv. Enhancing Technol. **2021**(2), 111–129 (2021)