# Testing Robustness of Homomorphically Encrypted Split Model LLMs

Lars Wolfgang Folkerts and Nektarios Georgios Tsoutsos

University of Delaware, Newark, DE 19716, USA
{folkerts,tsoutsos}@udel.edu

**Abstract.** Large language models (LLMs) have recently transformed many industries, enhancing content generation, customer service agents, data analysis and even software generation. These applications are often hosted on remote servers to protect the neural-network model IP; however, this raises concerns about the privacy of input queries. Fully Homomorphic Encryption (FHE), an encryption technique that allows for computations on private data, has been proposed as a solution to the challenge. Nevertheless, due to the increased size of LLMs and the computational overheads of FHE, today's practical FHE LLMs are implemented using a split model approach. Here, a user sends their FHE encrypted data to the server to run an encrypted attention head layer; then the server returns the result of the layer for the user to run the rest of the model locally. By employing this method, the server maintains part of their model IP, and the user still gets to perform private LLM inference. In this work, we evaluate the neural-network model IP protections of single layer split model LLMs, and demonstrate a novel attack vector that makes it easy for a user to extract the neural network model IP from the server, bypassing the claimed protections for encrypted computation. In our analysis, we demonstrate the feasibility of this attack, and discuss potential mitigations.

**Keywords:** Large Language Models · Model Extraction Attack · Fully Homomorphic Encryption · Split-Model Architecture

## 1 Introduction

Large language models (LLMs) have delivered transformative solutions for how we work, assisting with both daily tasks and automating complex analyses. Built on the transformer machine learning architecture [28], these models boast billions of parameters and are trained on vast datasets. As a result, they excel at text analysis and generation, providing efficient ways to perform unique data analytics, generate content, and author source code for novel applications [8,20,25].

Despite their numerous benefits, LLMs pose significant privacy risks. These models are often hosted on remote servers to protect the intellectual property (IP) of the underlying model [2]. Consequently, cloud servers receive input

queries from users, which may contain sensitive and private information [6,9,26]. In this scenario, a curious LLM provider could potentially breach user confidentiality by storing and analyzing these queries. Alternatively, LLMs can be run locally [22]; however, this requires model owners to open-source their models, resulting in a loss of control over their IP.

Recent research advancements have focused on developing Fully Homomorphic Encryption (FHE), a privacy-preserving form of cryptography, as a potential solution. FHE allows computations to be performed on encrypted data without needing to decrypt it first. In this approach, the user's query is encrypted and sent to the remote server. The remote server can then perform calculations on the encrypted queries without leaking any information about the underlying query. Finally, the encrypted result is sent back to the user, who can decrypt and analyze it.

Many FHE-based machine learning works focus on convolutional neural networks (CNNs) for image classification [1, 7, 15, 23]. One of the main libraries for FHE machine learning, ConcreteML [30], expanded upon the CNNs to offer support for many popular machine learning models, including boosted trees, neural networks, and large language models (LLMs). The LLM implemented as part of their library operates using a split-model approach. A split model typically divides the model into three segments. The user runs the first segment locally, then sends the first segment's output to the server to run the middle segment. Finally, the user receives the middle segment's output to finish the computation locally. This process is illustrated in Figure 1.

Split models have been proposed for training neural networks with the goal of providing some protection for crowdsourced training data [21,29]. In contrast, ConcreteML utilizes this approach for neural network inference. This method ensures user privacy by dividing the model into segments, where sensitive data is processed locally by the user before being sent to the server for further computation. In this scenario, the user's sensitive data is never fully exposed to the server, thus maintaining confidentiality. Additionally, this approach can limit the computational burden on the user's local device, as only a portion of the model is run locally. This balance between privacy and efficiency makes the split-model approach a promising solution for secure and scalable machine learning applications.

In this work, we investigate and evaluate the vulnerability of this approach to model extraction attacks in ConcreteML. In ConcreteML, only a small segment of the network, including the attention segment of a single layer, is encrypted. While this keeps the computational overhead low and guarantees user privacy, our work demonstrates that this does not necessarily ensure the protection of the data due to the high dimensionality of inputs and outputs. To highlight this issue, we develop a novel attack vector that allows users to extract the full model, bypassing the presumed guarantees of FHE computation.

**Roadmap:** The rest of the paper is organized as follows: Section 2 provides the necessary background on ConcereteML and the GPT2-Small Model, including their structures and functionalities. Section 3 details our overall methodology

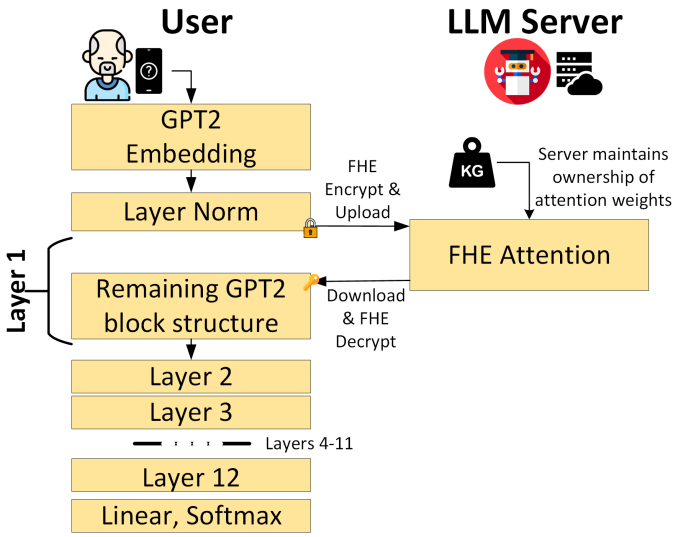


**Fig. 1. Split Model LLMs:** ConcreteML is using a Split Model approach for its large language model implementations. Here, a user runs attention of a single layer on the cloud, allowing for practical LLMs. In this work, we evaluate the model IP security of this approach, allowing us to steal the entire model.

and attack setup, describing how our exfiltration technique works. Section 4 presents our experimental results, along with our analysis of the attack's feasibility and potential mitigations. Finally, Section 5 discusses related works, and our concluding remarks are presented in Section 6.

## 2 Preliminaries

### 2.1 ConcreteML

#### 2.1.1 Privacy Preserving Machine Learning Model

The privacy-preserving machine learning model addresses the following scenario: a remote server (e.g., an LLM service provider) owns a trained machine learning model, which provides access to paid users. These users pay credits to upload their unique inputs and receive the generated outputs from the machine learning model. In this scenario, both the user and the machine learning server are assumed to be *honest but curious*. The machine learning server will carry out the computation but is incentivized to snoop on user data (e.g., for advertising purposes). Meanwhile, the user will upload correctly formed input data but may try to extract the machine learning algorithm parameters.

For user privacy protection, the ConcreteML library relies on the TFHE encryption scheme [4, 5]. TFHE is a fully homomorphic scheme, which means it

allows for basic computations such as additions, multiplications, and univariate activation functions on encrypted data, while still providing cryptographic hardness that makes it computationally infeasible for the server to learn information about the user's input. By default, ConcreteML parameterizes TFHE to provide 128-bits of cryptographic security.

For machine learning IP privacy, the machine learning server maintains ownership of all or part of the model IP, including the weights and biases. It is assumed by the ConcreteML library and other FHE works [1, 7, 13, 23] that this is sufficient protection for model IP.

### 2.1.2 Supported Algorithms

ConcreteML [30] is recognized as one of the leading open-source libraries for fully homomorphic encryption (FHE) in the context of machine learning inference. This innovative library leverages the capabilities of the scikit-learn machine learning library for plaintext training, providing users with access to a diverse array of models. Among these models are foundational algorithms such as linear regression and logistic regression, as well as more complex structures like decision trees and feed-forward neural networks. Notably, ConcreteML also includes implementations of attention mechanisms, expanding its utility for applications involving large language models and other architectures that benefit from attention-based techniques.

As already mentioned, ConcreteML operates using the TFHE encryption scheme [4, 5] for encrypted inference. Under the hood, this scheme is developed around lattice-based cryptography, utilizing the learning with errors problem as its foundational structure. TFHE operates on shortint data types, which consists of ciphertexts encoding bits of information [3]. To create a high-precision integer, multiple shortints are combined within a structured data format. However, encrypted floating-point operations are not supported in this encryption framework, presenting a challenge given the nature of training data typically utilized in scikit-learn.

To address this limitation, ConcreteML takes the trained scikit-learn models, along with several sample inputs, and compiles them into a custom FHE assembly code. This process includes converting the data into a high-precision fixed-point representation, effectively discretizing the input data based on heuristics derived from the sample inputs. Such discretization is crucial for maintaining the integrity of the model's performance when operating under the constraints of the TFHE scheme.

To mitigate potential discretization errors – particularly in deeper models, where such errors can significantly impact performance – it is necessary to retrain the discretized model. This fine-tuning process helps ensure that the model can adequately adjust to the fixed-point representation, optimizing its performance and reliability when deployed in a fully homomorphic encrypted environment. By following this approach, ConcreteML aims to preserve the efficacy of machine learning models while leveraging the security benefits of FHE.

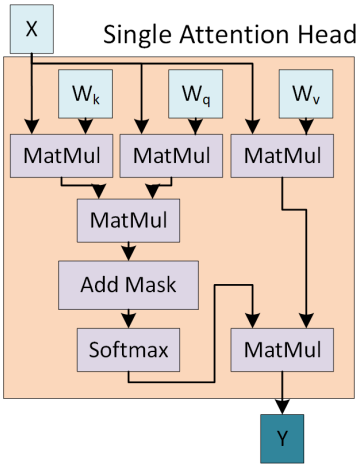


**Fig. 2. Single Attention Head:** ConcreteML encrypts a single attention head in its overall architechture. This diagram shows the encrypted operations.

### 2.2 GPT2-Small Model

The GPT2-Small [18,19] language model is constructed using 12 decoder blocks (Figure 3), or layers, that facilitate the processing of input data. Each of these blocks consists of multiple attention heads (Figure 2) [28], with additional layers in between to manage the gradient and normalize the data, further enhancing the model's capacity for learning complex patterns.

The initial step in this transformer architecture is the GPT2 embedding, which transforms words or tokens into a lower-dimensional numeric representation. For training, each word or token from the English dictionary (totaling 50,257 tokens for GPT2) is encoded as a one-hot vector $e$. A single-layer neural network is trained to convert this large one-hot representation into a compressed token embedding vector, which has a dimension of 768 for GPT2-Small. During plaintext inference, this embedding table functions as a lookup table, allowing the model to quickly retrieve the corresponding embedding for each token in the input sequence.

This token embedding process is applied to sequences of words, resulting in a tensor of dimensions $\mathbb{R}^{N \times D_e}$, where $N$ represents the length of the input sequence (i.e., the number of words in the user's input query) and $D_e$ denotes the dimension of the token embeddings. Following this, positional information is incorporated into the data to provide context regarding the order of tokens within the sequence. The final output, denoted as $X \in \mathbb{R}^{N \times D_e}$, serves as the input to the subsequent decoder layers, allowing the model to leverage both the semantic and positional aspects of the input data effectively.

Then the data is passed through the decoder layers. Each layer comprises several key components, as illustrated in Figure 3. The first component is the attention head. Each attention head processes the input $X$ to generate a key,

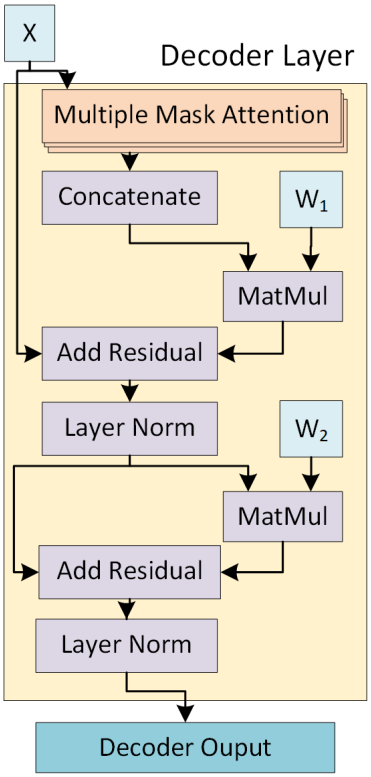


**Fig. 3. GPT2 Decoder Layer:** This diagram shows the entire GPT2 decoder layer. Only the attention segment is encrypted. $W_1$ and $W_2$ are weights learned in training (arbitrary subscripts); they become a linear layer through the subsequent matrix multiplication.

query, and value tuple via matrix multiplication, represented as:

$$K = XW_k \tag{1}$$

$$Q = XW_q \tag{2}$$

$$V = XW_v \tag{3}$$

Here, $K, Q, V \in \mathbb{R}^{N \times D_v}$, where $D_v$ denotes the dimensionality of the key, query, and value vectors. Once these vectors are obtained, the attention mechanism can be applied:

$$Attention = softmax(\frac{QK^T}{\sqrt{D_v}} + M)V. \tag{4}$$

In this equation, $M$ represents an optional mask that consists of large negative integers applied to any values that should be excluded from consideration during the attention calculation. This masking is particularly useful in scenarios where certain tokens in the sequence should not attend to others.

Several single-attention heads can operate in parallel, forming what is known as multi-head attention [28]. The outputs from all the individual attention heads are concatenated and processed through a subsequent linear layer. This allows the model to capture a richer set of relationships and dependencies within the input data, enhancing its overall ability to generate contextually relevant responses.

The remaining components of the GPT2 decoder block, as depicted in Figure 3, include residual connections (recurrent addition), layer normalization, and linear layers. After the multi-head attention mechanism, a residual connection is applied, where the input $X$ is added to the output of the attention layer. This addition helps facilitate better gradient flow during training, allowing the model to learn more effectively. Following this, layer normalization is applied to stabilize and accelerate training by normalizing the summed output. Next, a linear layer transforms the output from the attention mechanism. This transformation prepares the data for subsequent processing. Finally, a final residual connection and normalization are applied.

## 3 Methodology

### 3.1 ConcreteML's Model

ConcreteML presents a compelling use case example featuring GPT2-Small, a widely recognized open-source generative pretrained transformer language model. GPT2-Small is notable for its ability to generate coherent and contextually relevant text, making it a valuable tool for various applications in natural language processing. The architectural details of GPT2, including its multi-layer decoder transformer structure and the mechanisms that facilitate text generation, are discussed in depth in Section 2.2, while the dynamics of user-server interactions, which play a critical role in the deployment and security of the model, are elaborated in Section 2.1.1.

While model extraction attacks are a well-established concern in the realm of image classification, the application of such attacks to large language models (LLMs) presents a considerably greater challenge. This complexity arises from the substantial size and intricate architectures of LLMs, which often encompass billions of parameters and a multitude of layers. The extensive training data and the sophisticated mechanisms employed in these models make them less susceptible to straightforward extraction methods. Consequently, attackers face heightened difficulty in reverse-engineering the model's functionality or replicating its behavior, as they must navigate the vast parameter space and complex interactions within the architecture.

ConcreteML's implementation of GPT2-Small retains ownership of a single attention block, providing users access to the rest of the model, while restricting direct control over this critical component. This design choice effectively prevents users from running the language model independently; instead, they must depend on the server to perform computations of the missing attention layer homomorphically. As a result, the user has the ability to run a private LLM with

minimal computation cost, while the server maintains a level of IP protection via its ownership of the missing layer.

Since the cloud retains ownership of a part of the LLM, our key observations is that *the parameters of this attention head are the only thing required to create a counterfeit LLM.* The input and output vectors of this attention head are characterized by relatively high dimensional space compared to model extraction attacks, specifically $\mathbb{R}^{8\times768}$ for inputs and $\mathbb{R}^{8\times64}$ for outputs. This high dimensionality presents a potential vulnerability, as it can inadvertently leak considerable information about the underlying model parameters to the user, facilitating model extraction in our attack.

### 3.2 Our Attack Setup

In ConcreteML's design, the user has access to the majority of the model, allowing for a comprehensive examination of its architectural nuances. This access, combined with knowledge of the input and output dimensions of the encrypted segment, enables the user to infer details about the architecture in great detail. Moreover, given that many large language model (LLM) architectures, such as GPT2-Small, exhibit repetitive patterns, it is reasonable to assume that the user can infer the architectural design of the encrypted server component. For GPT2-Small, the encrypted segment is a single attention head, pictured in Figure 2.

With the architecture of the missing segment established, the objective of our attack shifts towards re-engineering a copy of the weights that remain under the control of the remote server. Specifically, these weights include $W_k$, $W_q$, and $W_v$, which correspond to the key, query, and value weights for the single attention head in GPT2-Small.

To achieve this, we train a *shadow model* that incorporates the attention head architecture, as shown in Figure 4. This shadow model serves as a proxy to approximate the behavior of the original GPT2 model on the server. For training data, we generate a dataset of random values for inputs, denoted as $X$, and submit these values as a set of encrypted queries to the GPT2 model. The corresponding outputs, denoted $y_{target}$, captured from the attention layer form input-output pairs, which correspond to the training set for our shadow model. Additional queries can be used for a validation set.

To train the shadow model effectively, it is also essential to select an appropriate loss function. Given that the primary objective is to replicate the weights of the missing attention head, robustness against outliers is less critical, and overfitting is allowed. In light of this, we have opted for the Mean Squared Error (MSE) loss function [12], defined as

$$Loss_{MSE} = \frac{1}{N_{out}} \sum_{i=1}^{N_{out}} (Attention(X) - y_{target})^2. \tag{5}$$

Here, $y_{target}$ represents the target output vector, and $N_{out}$ denotes the output vector dimensions. This formulation calculates the average of the squared differences between the predicted outputs from the attention mechanism and

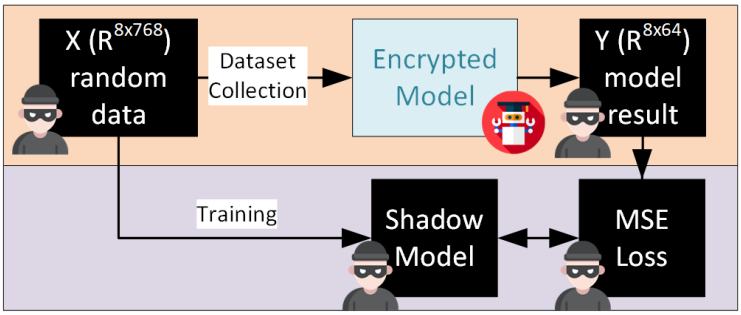


**Fig. 4. ConcreteML Model Extraction:** This figure shows an overview of our attack. The thief (black boxes) generates and sends a random query $X$ to the encrypted model on the remote server (blue box), and subsequently receives the result $Y$. This $(X, Y)$ dataset is used to train a shadow model with identical weights to the cloud model. The high dimensionality of inputs and outputs in ConcreteML's GPT2-Small model, as well as a limited encrypted segment, make this attack feasible.

the corresponding target values. By focusing on minimizing this loss, we can effectively train the shadow model to approximate the behavior of the attention head.

To optimize the training process, we have chosen the AdamW optimizer [14], a standard choice for gradient descent methods. AdamW incorporates adaptive learning rates and weight decay, which can enhance convergence speed and model generalization. This combination of the MSE loss function and the AdamW optimizer provides a solid foundation for training our shadow model, ultimately facilitating the extraction of the missing weights $W_k$, $W_q$, and $W_v$.

## 4 Results and Discussion

### 4.1 Experimental Setup

In our study, we assumed that the attacker possesses no prior knowledge of the dataset. To generate inputs for our model, we utilized a vector comprising uniformly random integers within the range of -63 to 32. These randomly generated inputs were subsequently encrypted wuth TFHE and transmitted to the remote model segment, from which we received the corresponding results. The input-output pairs obtained through this process constitute our dataset, which we utilized to train the shadow model. Additionally, we sent a total of one thousand queries to establish a validation dataset, which played a crucial role in assessing the efficacy of all training methodologies employed in our analysis.

Following the model training parameters discussed in the previous section, we conduct experiments with varying dataset sizes: specifically, one thousand, ten thousand, and fifty thousand queries (Figure 5). Our findings, shown in Figure 6, indicate that the model successfully trained on the provided data, achieving

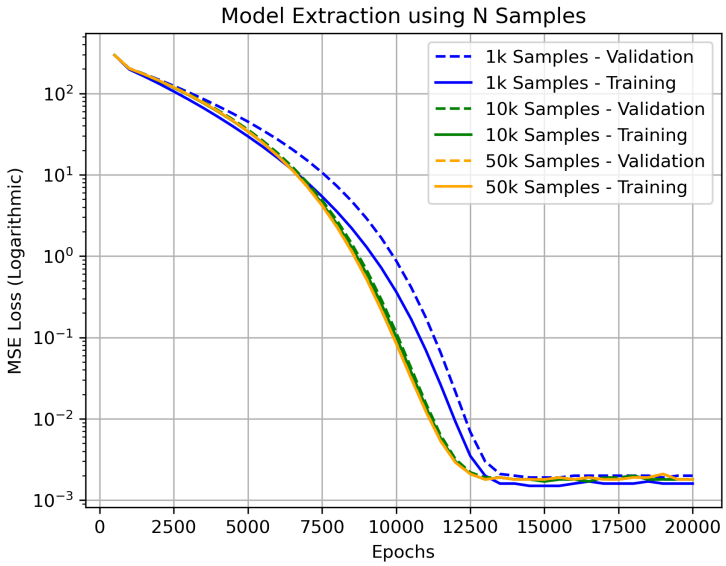


**Fig. 5. Model Extraction Attack:** This graph shows the training vs. validation MSE loss for our model extraction attack. We achieve a minimal loss function of 0.0018, indicating that the our trained Torch model closely matches the ConcreteML target model (minus discretization errors). This increases slightly to 0.0020 when the training set size is limited to 1000 samples, indicating a bound to where more samples would lead to a higher fidelity shadow model.

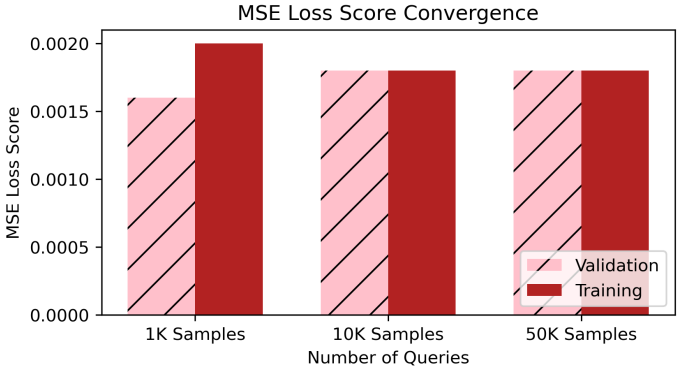


**Fig. 6. MSE Loss:** Mean square error loss function convergence. The training and validation set start diverging at 1k samples, indicating more samples are needed.

a mean square error (MSE) of 0.0018. This level of accuracy is sufficiently low to demonstrate the discretization error between the encrypted integer model and the floating-point shadow model. These results highlight the effectiveness of our approach and the potential for further improvements in model training and performance assessment.

### 4.2 Analysis of our Results

The utilization of random input data, along with our model's training designed to overfit, means that the training and validation sets perform similarly in the case of successful model extraction. Notably, in the experiments involving a large number of queries in the dataset, both the training and validation datasets exhibited a consistent downward trajectory during gradient descent, despite being independently distributed. This observation indicates that the model's weights were trending towards known values.

Conversely, in the attack scenario involving a dataset limited to one thousand queries, we observed that the validation set lagged behind the training set, despite eventually converging to the correct weights. This discrepancy indicates that the model encountered greater difficulty in predicting previously unseen information. Such a finding implies that we may be approaching the threshold of the minimum number of queries necessary for a successful dataset.

Regarding the limit of one thousand queries, this can be considered a reasonable size in the context of language models such as ChatGPT. In this framework, it is essential to note that one query corresponds to a single token, with the total token limit per prompt set at 4,096 tokens. Thus, a malicious user would be able to extract this key information under the guise of a single prompt.

### 4.3 Mitigations

There are two main factors that enable our model extraction attack. The first is the high dimensionality of input and output values, while the second is the size and complexity of the model.

Reducing the dimensionality of the internal layers of a high-dimensional LLM is challenging without compromising model performance. However, we expect this issue to become more challenging as the GPT model size grows. For instance, increasing the model size from GPT-Small to GPT-ExtraLarge increases the token embedding size from 768 to 1600, providing malicious users with more information.

To enhance robustness against model extraction attacks, it is important to recognize that the current research on extracting entire LLMs is still ongoing, due to their inherent complexity. To adequately protect its data, existing models, such as the one in ConcreteML, must incorporate more complexity and a higher number of parameters in their encrypted segment than a single attention head.

To achieve this, one potential approach would be to implement an efficient LayerNorm feature and add it to the target framework (ConcreteML in our case). This LayerNorm function would facilitate the analysis of multiple layers within the model, thereby increasing its capacity to learn and process more intricate patterns in the data. By integrating LayerNorm, the target framework can enhance its training stability and performance, enabling the model to effectively manage the interactions between various layers while preserving critical information.

Implementing multiple, subsequent attention heads across layers would allow the model to capture a broader range of dependencies and relationships within the input data. This multi-head approach can provide a richer representation and make it more challenging for potential attackers to extract sensitive information.

## 5 Related Work

Model extraction attacks have been extensively studied in the context of image classification using convolutional neural networks (CNNs). The core idea is to collect sufficient data samples by querying the model, enabling attackers to replicate its classification results. However, these previous works often require a significantly higher number of queries, as the label-only classification outputs provide limited information to the user. This limitation means that extracting a model's internal behavior accurately necessitates more extensive querying, which can be resource-intensive.

Papernot et al. [17] conducted one of the first studies on model extraction attacks using synthetic data based on a small subset of real MNIST digit images. They achieved a modest accuracy of 81.20% with 6,400 queries. Building on this, Juuti et al. [10] increased the number of queries to 102,400, resulting in a significant improvement to 97.9% accuracy. Both studies innovated by generating synthetic datasets of MNIST digits that closely mimic real numbers.

CIFAR-10 has also been a target for model extraction studies. For instance, Shokri et al. [24] developed a membership inference attack, achieving about 60% test set accuracy with 15,000 queries. Truong et al. [27] tackled the more complex CIFAR-10 dataset by utilizing surrogate images from the CIFAR-100 dataset, reaching an accuracy of 88.1%, but at the cost of requiring an extensive 20 million queries. These previous studies, which primarily focus on image classification tasks, often require a substantially higher number of queries to achieve high-fidelity model extraction. For instance, methods involving thousands or even millions of queries may yield only modest accuracy improvements, making the process both resource-intensive and time-consuming.

These previous studies, which primarily focus on image classification tasks, often require a substantially higher number of queries to achieve high-fidelity model extraction. For instance, methods involving thousands or even millions of queries may yield only modest accuracy improvements, making the process both resource-intensive and time-consuming.

Conversely, our work represents a significant advancement in this domain. We achieve high-quality model extraction with just 1,000 queries, a remarkable reduction that enhances both efficiency and practicality. Our unique focus on stealing the exact weights of the model—rather than simply aiming for correct classifications—enables us to leverage the mean squared error loss function. This choice facilitates quick convergence within our limited query set, allowing us to extract model parameters with impressive precision.

Moreover, we are among the first to apply model extraction techniques to large language models, an area that has been relatively underexplored in the

literature. By addressing the specific challenges posed by LLMs, our research not only expands the scope of model extraction but also uncovers critical vulnerabilities in this vital field of machine learning.

Finally, several related studies propose techniques such as prediction poisoning or adaptive misinformation to mitigate malicious inputs and prevent model extraction attacks [11, 16]. These attacks primarily target image classification systems; when an out-of-band image is provided (i.e., a random malicious query), the cloud service intentionally generates an irrelevant response. However, for ConcreteML's application, implementing such defenses is considerably more challenging. Firstly, the encrypted segment resides within an intermediate layer, complicating the identification of out-of-band information. Secondly, any out-of-band detection algorithm must be executed homomorphically, limiting the types of algorithms available. Finally, the availability of text data for training large language models (LLMs) is significantly greater than that of image data, rendering this defense against synthetic data less effective.

## 6 Conclusion

In this paper, we explore the vulnerability of the split-model approach in ConcreteML to model extraction attacks. Our goal is to assess the security implications of using TFHE in protecting user data during neural network inference. By generating random input data and training a shadow model, we are able to demonstrate that it is possible to extract the full model, highlighting a significant privacy risk. Our experiments reveal that even with a limited dataset of one thousand queries, the model could eventually converge to the correct weights, demonstrating a successful model extraction scenario. This finding underscores the importance of addressing the high dimensionality of input and output values and the complexity of the model to enhance security.

To mitigate these risks, we propose incorporating more complexity and a higher number of parameters in the encrypted segment of the model. Implementing an efficient LayerNorm feature and multiple attention heads across layers can improve the model's robustness against extraction attacks. These enhancements will enable the model to capture a broader range of dependencies and relationships within the input data, making it more challenging for attackers to extract sensitive information. Our work emphasizes the need for ongoing research and development to ensure the security and privacy of machine learning models in practical applications.

## Acknowledgments

L. Folkerts and N.G. Tsoutsos would like to acknowledge the support of the National Science Foundation (Award 2239334).

## References


1. Ehud Aharoni, Allon Adir, Moran Baruch, Nir Drucker, Gilad Ezov, Ariel Farkash, Lev Greenberg, Ramy Masalha, Guy Moshkowich, Dov Murik, et al. Helayers: A tile tensors framework for large neural networks on encrypted data. *arXiv preprint arXiv:2011.01805*, 2020.
2. Jawid Ahmad Baktash and Mursal Dawodi. Gpt-4: A review on advancements and opportunities in natural language processing. *arXiv preprint arXiv:2305.03195*, 2023.
3. Ilaria Chillotti. Tfhe deep dive - part i - ciphertext types, May 2022.
4. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. Tfhe: fast fully homomorphic encryption over the torus. *Journal of Cryptology*, 33(1):34–91, 2020.
5. Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. TFHE: Fast fully homomorphic encryption library, August 2016. https://tfhe.github.io/tfhe/.
6. Adam Derose. These companies have banned or limited ChatGPT at work. *Morning Brew*, May 2023.
7. Lars Folkerts, Charles Gouert, and Nektarios Georgios Tsoutsos. Redsec: Running encrypted discretized neural networks in seconds. *Cryptology ePrint Archive*, 2021.
8. Jorge Jinchuña Huallpa et al. Exploring the ethical considerations of using chat gpt in university education. *Periodicals of Engineering and Natural Sciences*, 11(4):105–115, 2023.
9. JaxonAI. Companies that have banned ChatGPT, Jun 2023.
10. Mika Juuti, Sebastian Szyller, Samuel Marchal, and N Asokan. Prada: protecting against dnn model stealing attacks. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 512–527. IEEE, 2019.
11. Sanjay Kariyappa and Moinuddin K Qureshi. Defending against model stealing attacks with adaptive misinformation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2020.
12. Taehyeon Kim, Jaehoon Oh, NakYil Kim, Sangwook Cho, and Se-Young Yun. Comparing kullback-leibler divergence and mean squared error loss in knowledge distillation. *arXiv preprint arXiv:2105.08919*, 2021.
13. Joon-Woo Lee, HyungChul Kang, Yongwoo Lee, Woosuk Choi, Jieun Eom, Maxim Deryabin, Eunsang Lee, Junghyun Lee, Donghoon Yoo, Young-Sik Kim, et al. Privacy-preserving machine learning with fully homomorphic encryption for deep neural network. *IEEE Access*, 10:30039–30054, 2022.
14. I Loshchilov. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.
15. Qian Lou and Lei Jiang. SHE: A Fast and Accurate Deep Neural Network for Encrypted Data. *Advances in Neural Information Processing Systems*, 32:10035–10043, 2019.
16. Tribhuvanesh Orekondy, Bernt Schiele, and Mario Fritz. Prediction poisoning: Towards defenses against dnn model stealing attacks. *arXiv preprint arXiv:1906.10908*, 2019.
17. Nicolas Papernot, Patrick McDaniel, Ian Goodfellow, Somesh Jha, Z Berkay Celik, and Ananthram Swami. Practical black-box attacks against machine learning. In *Proceedings of the 2017 ACM on Asia conference on computer and communications security*, pages 506–519, 2017.

18. Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.
19. Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.
20. Partha Pratim Ray. Chatgpt: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope. *Internet of Things and Cyber-Physical Systems*, 2023.
21. Daniele Romanini, Adam James Hall, Pavlos Papadopoulos, Tom Titcombe, Abbas Ismail, Tudor Cebere, Robert Sandmann, Robin Roehm, and Michael A Hoeh. Pyvertical: A vertical federated learning framework for multi-headed splitnn. *arXiv preprint arXiv:2104.00489*, 2021.
22. Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.
23. Amartya Sanyal, Matt Kusner, Adria Gascon, and Varun Kanade. TAPAS: Tricks to accelerate (encrypted) prediction as a service. In *International Conference on Machine Learning*, pages 4490–4499. PMLR, 2018.
24. Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE symposium on security and privacy (SP)*, pages 3–18. IEEE, 2017.
25. Tristen Taylor. The top types of ai-generated content in marketing, Oct 2023.
26. Taylor Telford and Pranshu Verma. Employees want ChatGPT at work. Bosses worry they'll spill secrets. *The Washington Post*, Jul 2023.
27. Jean-Baptiste Truong, Pratyush Maini, Robert J Walls, and Nicolas Papernot. Data-free model extraction. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 4771–4780, 2021.
28. A Vaswani. Attention is all you need. *Advances in Neural Information Processing Systems*, 2017.
29. Praneeth Vepakomma, Otkrist Gupta, Tristan Swedish, and Ramesh Raskar. Split learning for health: Distributed deep learning without sharing raw patient data. *arXiv preprint arXiv:1812.00564*, 2018.
30. Zama. Concrete ML: a privacy-preserving machine learning library using fully homomorphic encryption for data scientists, 2022. `https://github.com/zama-ai/concrete-ml`.