Robust and Efficient Watermarking of Large Language Models Using Error Correction Codes

Xiaokun Luan
Peking University
Beijing, China
luanxiaokun@pku.edu.cn

Zeming Wei
Peking University
Beijing, China
weizeming@stu.pku.edu.cn

Yihao Zhang
Peking University
Beijing, China
zhangyihao@stu.pku.edu.cn

Meng Sun
Peking University
Beijing, China
sunm@pku.edu.cn

Abstract

Large language models (LLMs) have demonstrated remarkable performance in various tasks, but protecting their intellectual property (IP) is a growing concern. While conventional training-based watermarking is computationally expensive, function invariant transformations (FITs) offer a lightweight alternative, yet remain vulnerable to adaptive attacks. We propose a novel white-box watermarking scheme combining error correction codes (ECCs) with weight permutations. By encoding model identifiers using ECCs, our approach guarantees reliable watermark extraction under various attacks. A linear assignment-based extraction algorithm further enhances its efficiency. Extensive evaluations show that our method offers robust watermarking capabilities. It has a minimal impact on model performance while effectively defending against removal and forgery attacks. Overall, our approach serves as a scalable and reliable solution for safeguarding the intellectual property of LLMs.

Keywords

watermarking, error correction code, large language models

1 Introduction

The emergence of large language models has brought about a remarkable transformation in the field of machine learning, especially within the domain of natural language processing. Noteworthy examples such as GPT-4 [44], Llama [63], and DeepSeek [15] have exhibited impressive performance in tasks ranging from natural language understanding [16, 61], code generation [4, 43], to complex reasoning [67, 74]. Their widespread adoption has driven significant advancements in the field.

However, the high performance comes at a significant cost. Training large language models requires substantial computational resources and expertise, making their development highly expensive. As these models grow in value, protecting their intellectual property (IP) becomes increasingly important. Instances of IP infringement have occurred [50], where pre-trained models developed by one party were falsely claimed as the original work of another. Such incidents highlight the urgent need for robust mechanisms to safeguard model ownership and authenticity. This concern is further strengthened by recent legislative efforts worldwide. Governments in the United States, the European Union, and China have introduced AI-related regulations [12, 17, 62] to oversee the development and use of AI technologies. In this context, watermarking techniques [13, 25, 69] have emerged as a promising approach to address these challenges by embedding unique identifiers into models to establish ownership and trace provenance. Traditional neural network watermarking methods typically rely on optimization-based techniques to embed watermarks during pre-training or fine-tuning phases. These approaches have shown success in smaller neural networks, effectively balancing utility and robustness. However, the computational overhead of such training-based watermarking methods becomes prohibitive as the scale of models grows, making them impractical for large language models. Tackling these challenges necessitates the development of scalable watermarking techniques that can efficiently embed and robustly verify watermarks without significantly affecting model performance.

A promising alternative to training-based watermarking is leveraging function invariant transformations [18] (FITs), which embed watermarks without modifying the model architecture or training process. FITs preserve a model's functionality while altering its internal representation, such as by permuting the dimensions of the model weights. The different parameters of FITs function as unique watermarks, enabling model providers to trace the model's provenance without the need for retraining. While the FIT-based watermarking method offers a lightweight and efficient approach to embedding watermarks in large language models, it typically necessitates white-box access for extraction, and most importantly, its robustness against adaptive attacks remains a significant challenge. Adversaries with knowledge of the watermarking scheme can apply FITs with random parameters to remove or forge the watermark, rendering FIT-based watermarks vulnerable to tampering and unauthorized modifications. Moreover, the existing method for extracting weight permutation transformations is computationally expensive, causing a significant performance overhead during watermark verification.

To address these challenges, we propose a novel white-box watermarking scheme that integrates error correction codes (ECCs) with weight permutations to efficiently and robustly watermark large

Proceedings on Privacy Enhancing Technologies 2025(4), 183–200

https://doi.org/10.56553/popets-2025-0126

language models. Weight permutation, as a type of FIT, reorders the dimensions of transformer model weights while preserving functionality. It enables the generation of functionally equivalent copies of the original model with distinct weight arrangements, where the different transformation parameters (*i.e.*, permutations) serve as unique watermarks. To enhance the robustness of this simple FIT-based watermarking approach, we encode the model identifier using generalized Reed-Solomon codes, a type of ECC, thereby transforming it into a codeword with error correction capabilities. Through a sophisticated permutation mapping design, the encoded model identifier is subsequently converted to a sequence of permutations, which acts as the watermark embedded within the model weights. This approach enables model providers to almost always detect tampering attempts during watermark extraction, significantly increasing the difficulty of adaptive attacks. Moreover, we introduce a linear assignment-based permutation extraction algorithm that efficiently retrieves watermarks while resisting watermark obfuscation attacks leveraging other FITs. Our approach is extensively evaluated on six large language models, demonstrating its effectiveness in inserting and extracting watermarks with minimal impact on model performance, while offering robust protection against removal and forgery attacks and various model modifications.

Overall, our contributions can be summarized as follows:

(1) We propose a novel white-box watermarking scheme that integrates error correction codes with weight permutations, providing theoretical guarantees for detecting tampering attempts and ensuring reliable watermark recovery.

(2) For efficient watermark extraction, we devise a permutation extraction algorithm based on linear assignment solving, which can resist watermark obfuscation attacks based on other function invariant transformations.

(3) Our extensive evaluation demonstrates that our approach has minimal impact on large language model performance and provides strong resistance against removal and forgery attacks and various model modifications.

2 Preliminaries

In this section, we first provide a brief overview of the transformer architecture, which is the foundation of large language models. We then describe the background of function invariant transformations (FITs) and generalized Reed-Solomon codes.

2.1 Transformer-based Language Models

The input of a transformer neural network is a sequence of tokens $(x_1, \cdots, x_n) \in \mathcal{V}^n$. An embedding layer $E \in \mathbb{R}^{|\mathcal{V}| \times d}$ maps each token x_i to a vector $z_i \in \mathbb{R}^d$, where $|\mathcal{V}|$ is the vocabulary size and d is the *embedding dimension*. The input sequence $\boldsymbol{z} = (z_1, \cdots, z_n) \in \mathbb{R}^{n \times d}$ is passed through a series of blocks, each consisting of a self-attention layer and a feed-forward network, and the output is a sequence of hidden states $\boldsymbol{h} = (h_1, \cdots, h_n) \in \mathbb{R}^{n \times d}$. In the following, we describe the relevant components in the transformer architecture that are commonly used in large language models.

2.1.1 Self-attention Layers. In a self-attention layer, the input sequence $\boldsymbol{z}$ is transformed into queries $Q = \boldsymbol{z}W_q \in \mathbb{R}^{n \times d_k}$, keys $K = \boldsymbol{z}W_k \in \mathbb{R}^{n \times d_k}$, and values $V = \boldsymbol{z}W_v \in \mathbb{R}^{n \times d_v}$, where W_q, W_k, and W_v are learnable weights, and we refer to them as query, key, and value weights, respectively. The output is computed as a weighted sum of the values:

$$\text{Attention}\,(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V, \tag{1}$$

where Softmax is applied row-wise. The attention operator is applied multiple times in parallel, resulting in h output *heads* that are concatenated and projected back to the original dimension:

$$\text{MultiHead}\,(\boldsymbol{z}) = \text{Concat}\,(\text{head}_1, \cdots, \text{head}_h)\,W_o, \tag{2}$$

where $\text{head}_i = \text{Attention}(\boldsymbol{z}W_q^{(i)}, \boldsymbol{z}W_k^{(i)}, \boldsymbol{z}W_v^{(i)})$, and $W_o \in \mathbb{R}^{hd_v \times d}$ is the learnable output projection matrix. In practice, d_k and d_v are usually set equal, and the query weights $W_q^{(i)}$ are stacked into a single matrix $W_q \in \mathbb{R}^{d \times hd_k}$ and similarly for W_k and W_v to enable parallel computation. We call d_k the *hidden dimension* of the attention head.

Many recently developed large language models adopt Grouped-Query Attention (GQA) [3] to improve computation efficiency. Instead of using independent attention heads, GQA divides h_q query heads into h_k groups, where each group shares a single key and value head. Suppose there are g query heads in each group, then $h_q = g \cdot h_k$. The i-th key head $K^{(i)}$ is paired with g consecutive query heads $Q^{(j)}$ for $i \cdot g \le j < (i+1) \cdot g$. The stacked query, key, and value weights have shape $d \times h_q d_k$, $d \times h_k d_k$, and $d \times h_k d_v$, respectively.

2.1.2 Feed-forward Layers and Residual Connections. The output of attention is then fed to a feed-forward network consisting of two linear layers with an activation function Act:

$$\text{FFN}\,(\boldsymbol{h}) = \text{Act}\,(\boldsymbol{h}W_1 + b_1)\,W_2 + b_2, \tag{3}$$

where $W_1 \in \mathbb{R}^{d \times d_{\text{ff}}}$, $b_1 \in \mathbb{R}^{d_{\text{ff}}}$, $W_2 \in \mathbb{R}^{d_{\text{ff}} \times d}$, and $b_2 \in \mathbb{R}^d$ are learnable parameters, and common choices for activation functions include ReLU, GELU [26], and SwiGLU [55]. d_{ff} is usually called the *intermediate dimension* of the feed-forward network.

Residual connections are applied for each self-attention layer and feed-forward network:

$$\boldsymbol{h}^l = \text{ATN}^l\left(\text{LN}^l_{\text{atn}}(\boldsymbol{z}^l)\right) + \boldsymbol{z}^l, \tag{4}$$

$$\boldsymbol{z}^{l+1} = \text{FFN}^l\left(\text{LN}^l_{\text{ffn}}(\boldsymbol{h}^l)\right) + \boldsymbol{h}^l, \tag{5}$$

where LN is the normalization operator, such as Layer Normalization [5] or RMSNorm [76], and ATN and FFN are the self-attention layer and feed-forward network, respectively.

2.1.3 Positional Embeddings. Positional embeddings are used to encode the position of tokens in the input sequence. Some relative positional embeddings may change Eq. (1). For example, the widely used rotary embeddings [59] compute the inner product of the m-th query and the n-th key as:

$$Q_m K_n^\top = z_m W_q R_{\Theta, n-m}\,(z_n W_k)^\top, \tag{6}$$

where $R_{\Theta,n}$ is a block diagonal matrix with 2×2 rotation matrices:

$$\left(R_{\Theta,n}\right)_i = \begin{pmatrix} \cos n\theta_i & -\sin n\theta_i \\ \sin n\theta_i & \cos n\theta_i \end{pmatrix}, \tag{7}$$

with rotation angles $\theta_i = 10,000^{-2i/d}$.

2.1.4 Language Modeling Head. The last hidden state of the transformer is passed through a linear layer to generate logits for each token in the vocabulary:

$$\text{Logits}(\boldsymbol{h}) = \boldsymbol{h}W_{\text{lm}} + b_{\text{lm}}. \tag{8}$$

Some models use weight tying [49] to share the embedding matrix E and the output projection matrix W_{lm} by setting $W_{\text{lm}} = E^\top$.

2.2 Function Invariant Transformations

Function invariant transformations can generate functionally equivalent copies of the model by leveraging the model's invariance to certain operations. The operations applied to the model are regarded as a stealthy watermark, which can be extracted (under a non-blind white-box setting) and verified by the watermark owner. For transformer neural networks, three types of operations are proposed to generate functionally equivalent models in [18]: weight permutation, scaling/unscaling, and invertible matrices in QK products, which are briefly described as follows.

Weight permutation (WP) can be applied at four different places in the transformer neural network:

(1) *Embedding dimension.* The columns of the embedding matrix E can be permuted, and invariance can be obtained by propagating the permutation to all other parameters.
(2) *Intermediate dimension.* The intermediate output between two linear layers in the feed-forward network can be permuted while preserving functionality.
(3) *Attention heads.* The heads of multi-head attention are interchangeable, as long as the projection matrices are permuted accordingly. For GQA, the groups of query heads with their paired key/value heads can be permuted, and the arrangement of query heads within each group can be arbitrary.
(4) *Hidden dimension.* Each attention head can have its hidden dimension permuted if it does not affect the attention mechanism in Eq. (1). This is not the case for rotary embeddings, where the permuted hidden dimension does not preserve Eq. (6).

Scaling/unscaling (SU) operation multiplies the layer normalization parameters by a constant vector α, and divides the rows of its following (preceding) linear layer by the same vector to remain functionally equivalent.

Invertible matrix in query-key products (IM) operation multiplies the query weights by an invertible matrix and the key weights by its inverse in the query-key product. To remain function invariant, it is required that the result of Eq. (1) is not affected by the transformation. In fact, hidden dimension permutation is a special case of the invertible matrix in query-key products, as permutation matrices are orthogonal and thus invertible.

We will revisit these transformations and provide their mathematical formulations in the context of watermarking in Section 5.

2.3 Generalized Reed-Solomon Codes

To transmit data over unreliable channels, error correction codes are used to encode the message in a redundant way, allowing the receiver to detect and correct errors. In this work, we use generalized Reed-Solomon codes, which are a class of non-binary linear error correction codes. A generalized Reed-Solomon code $\text{GRS}_{n,k}(\boldsymbol{e}, \boldsymbol{v})$ is defined over a finite field GF of size q, where q is a prime power, $\boldsymbol{e}$ and $\boldsymbol{v}$ are evaluation points and scaling factors, and k and n are the message and codeword lengths ($k < n < q$), respectively. In the following, we use $\text{GRS}_{n,k}$ to denote a generalized Reed-Solomon code with parameters n and k.

A message $msg \in \text{GF}^k$ is a sequence consisting of k symbols in the finite field, and it is encoded into a codeword of n symbols, denoted as $c \in \text{GF}^n$. The received codeword r is the codeword c with errors and erasures, where an erasure is a corrupted symbol with known position, and an error is a corrupted symbol with unknown position. Any combination of erasures and errors can be corrected by $\text{GRS}_{n,k}$ as long as the condition

$$2E + S \leq n - k \tag{9}$$

holds, where E is the number of errors and S is the number of erasures. Therefore, $n - k$ is called the *error-correction capability* of the generalized Reed-Solomon code $\text{GRS}_{n,k}$. When the number of erasures and errors exceeds the error-correction capability (*i.e.*, $2E + S > n - k$), $\text{GRS}_{n,k}$ either fails to decode or returns an incorrect message. We use GRS_{ENC} and GRS_{DEC} to denote the encoding and decoding process of a generalized Reed-Solomon code when the finite field and other parameters are clear from the context,*i.e.*,

$$\text{GRS}_{\text{ENC}} : \text{GF}^k \to \text{GF}^n, \tag{10}$$

$$\text{GRS}_{\text{DEC}} : \text{GF}^n \rightharpoonup \text{GF}^k, \tag{11}$$

where $\rightharpoonup$ denotes a partial mapping that may be undefined for some inputs. Although generalized Reed-Solomon codes are not designed for encryption, it is computationally hard to recover the original message without the full knowledge of the code parameters, especially when the finite field is very large.

3 Threat Model

We consider a threat model involving two primary parties: the model provider and users. The model provider is the legitimate owner of the LLM who wishes to embed watermarks into the model to establish ownership and enable provenance tracing. The watermarked models are distributed to users, who may be either non-malicious users or adversaries. All users, whether non-malicious or adversarial, have full white-box access to the watermarked model, including the ability to inspect and modify the model weights, but they do not have access to the original, non-watermarked model. An adversary's objective is to remove or forge the embedded watermark to evade detection, while still preserving the model's utility. We assume the adversary may know the watermarking scheme, but not the secrets used by the provider during watermarking. Watermark extraction and verification is performed by the model provider, who requires white-box access to the watermarked and original models.

4 Overview

Fig. 1 provides an overview of the integration of error correction codes with weight permutations for watermarking large language models. Given a model identifier and a large language model, the proposed method first encodes the identifier into a codeword with a generalized Reed-Solomon (GRS) code, which is then converted to a sequence of permutations. Weights in the model are permuted according to the permutations, resulting in a watermarked model.

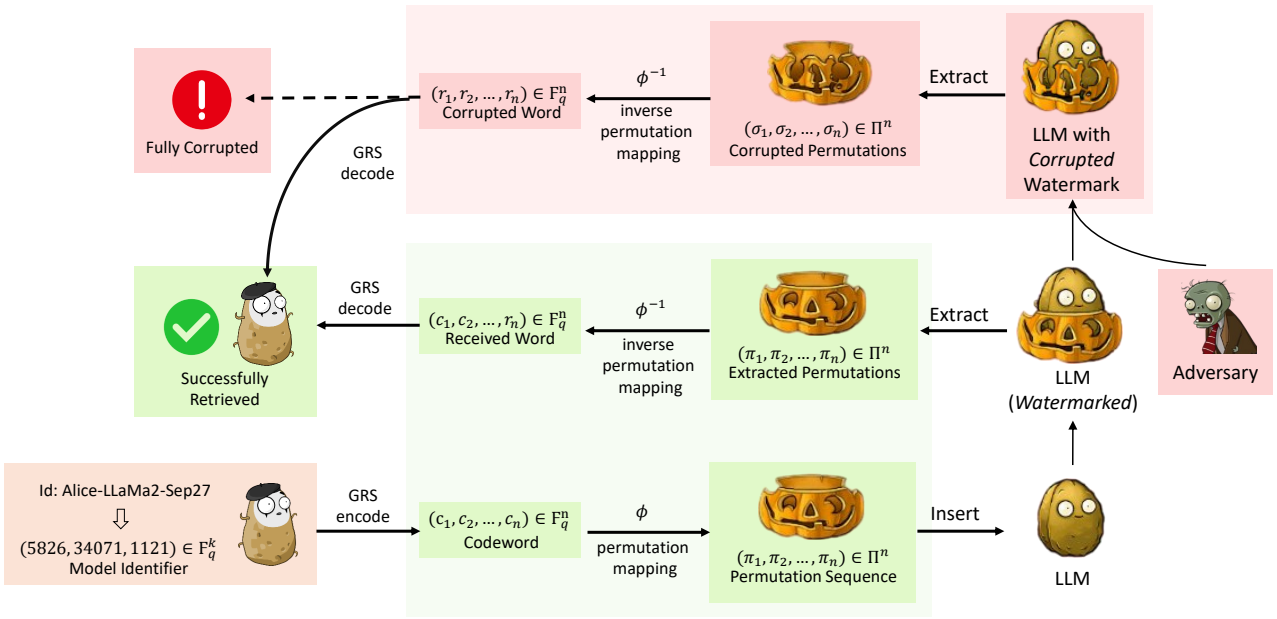

Figure 1: An overview of the integration of error correction codes with weight permutations for watermarking large language models.

To retrieve the model identifier, permutations are extracted from the watermarked model and then decoded using the GRS code and the permutation mapping. The usage of error correction codes ensures that the model identifier can be reliably recovered from the extracted permutations, as long as the number of corrupted permutations is within its error correction capability.

In Section 5, we first describe the permutation insertion process, which is responsible for reordering weights in the model while preserving its functionality. We extend the WP transformation proposed in [18] to support the commonly used GQA mechanism. Moreover, we introduce a linear assignment-based permutation extraction algorithm to efficiently extract permutations from the model. These two procedures alone cannot defend against adaptive attacks, which are also discussed in Section 5.

In Section 6, we present the design of the permutation mapping to encode symbols in the model identifier into permutations and decode permutations back into symbols. Combined with the generalized Reed-Solomon code, it can detect corrupted permutations during extraction and correct them to recover the original model identifier. We also propose an enhancement to the permutation extraction algorithm to defend against watermark obfuscation attacks based on SU and IM transformations.

5 Weight Permutation Watermarking

We first introduce a weight permutation watermarking scheme for large language models without using error correction codes. The main difference between this scheme and the vanilla FIT-based watermarking method [18] is that only weight permutations are involved, and it additionally supports grouped-query attention, which is not covered in the vanilla FIT-based method. We also propose a linear assignment-based permutation extraction algorithm to efficiently extract watermarks, which has a better time complexity than the brute-force search used in previous studies. Finally, we discuss potential attacks on this watermarking scheme. In Section 6, we will extend this scheme by combining error correction codes with weight permutations to enhance the watermark robustness.

5.1 Permutation Insertion

There are four types of weight permutation transformations that maintain model invariance, as described in Section 2. We denote them as T_{emb}, T_{ffn}, T_{heads}, and T_{hid}, which correspond to embedding dimension, intermediate dimension, attention heads, and hidden dimension, respectively. We define a weight permutation transformation as $\textsc{Perm}(l, t, \pi)$, where π is a permutation applied to the weights in the l-th attention block, in a way specified by permutation type $t \in \mathcal{T} = \{T_{\text{emb}}, T_{\text{ffn}}, T_{\text{heads}}, T_{\text{hid}}\}$, and other weights are adjusted accordingly to ensure the same functionality.

We refer to a certain layer in the model as a *permutation slot* if it can be applied with a weight permutation transformation, defined by a block index l, a permutation type $t \in \mathcal{T}$, and a set of permissible permutations R that preserve function invariance, and we denote the sequence of permutation slots of a model f as $S_f = [s_1, \cdots, s_n]$. The number of permutation slots is determined by the model structure and varies for different models. For example, Llama3.2-1B has 16 attention blocks, where each block has two slots[1] for inserting permutations, so $n \leq 2 \times 16 + 1 = 33$ (one extra slot in the embedding layer). The permutation insertion process involves applying weight permutation transformations $\{\textsc{Perm}(l_i, t_i, \pi_i)\}_{i=1}^n$ to the model f at each slot s_i such that $\pi_i \in R_i$. The permutation

[1]Llama models use rotary embeddings, so T_{hid} is not applicable.

sequence $[\pi_1, \cdots, \pi_n]$ is regarded as a unique watermark of the watermarked model f'.

We use Π_d to denote the set of all permutations of d elements, and we use $\pi^k \in \Pi_d^k \subseteq \Pi_{dk}$ to denote a block permutation of d blocks with k consecutive elements in each block. Specifically,

$$\pi^k(k \cdot i - j) = k \cdot \pi(i) - j \text{ for } 1 \le i \le d, 0 \le j < k. \quad (12)$$

i.e., π^k acts on the blocks as whole units while preserving the relative positions of elements within each block (*e.g.*, $(3, 4, 1, 2) \in \Pi_2^2$). We use $\mathrm{BP}(\pi; k) := \pi^k$ to denote the block permutation induced by π with block size k. When applying a permutation π to a weight matrix W, we use $W_{:,\pi}$ (resp. $W_{\pi,:}$) to denote the matrix obtained by permuting the columns (resp. rows) of W according to π. We use R to denote the set of permissible permutations for a specific permutation type.

5.1.1 Embedding Dimension. The operation $\textsc{Perm}(\pi, \cdot, T_{\text{emb}})$ permutes the columns of the embedding matrix $E \in \mathbb{R}^{|\mathcal{V}| \times d}$ according to $\pi \in R_{\text{emb}} = \Pi_d$, and propagates the permutation to other parameters accordingly, *i.e.*, $E' = E_{:,\pi}$, $W_q{}' = (W_q)_{\pi,:}$ (similarly for all W_k, W_v, W_1, b_2, LN, and W_{lm}[2]), and $W_o{}' = (W_o)_{:,\pi}$ (similarly for all W_2).

5.1.2 Intermediate Dimension. $\textsc{Perm}(\pi, l, T_{\text{ffn}})$ applies $\pi \in R_{\text{ff}} = \Pi_{d_{\text{ff}}}$ to the output dimension of the first linear layer in the l-th feed-forward network, and permutes the input dimension of the second linear layer, *i.e.*, $W_1{}' = (W_1)_{:,\pi}$, $b_2{}' = (b_2)_\pi$, and $W_2{}' = (W_2)_{\pi,:}$.

5.1.3 Attention Heads. $\textsc{Perm}(\pi, l, T_{\text{heads}})$ rearranges heads at the l-th self-attention layer. Depending on whether GQA is used, the permissible permutations and the insertion operation are different.

If the numbers of query heads and key/value heads are the same, *i.e.*, GQA is not used, the permutation π is applied on these heads, thus should be chosen from $R_{\text{heads}} = \Pi_h$ where h is the number of heads. The query, key, value, and projection weights are permuted according to the induced block permutation $\pi^{d_k} \in \Pi_h^{d_k}$:

$$\begin{aligned} W_q{}' &= (W_q)_{:,\pi^{d_k}}, \quad W_k{}' = (W_k)_{:,\pi^{d_k}}, \\ W_v{}' &= (W_v)_{:,\pi^{d_k}}, \quad W_o{}' = (W_o)_{\pi^{d_k},:}. \end{aligned} \quad (13)$$

If GQA is employed with $h_q = g \cdot h_k$ ($g > 1$), the query heads and key/value heads should be permuted differently. The permutation on key/value heads can be arbitrarily chosen as $\pi \in \Pi_{h_k}$, and the groups of query heads are permuted accordingly. On the other hand, each group of query heads can be independently permuted. For example, if $g = 3$, $h_k = 2$, and $\pi = (2, 1)$, then the two query groups are swapped, and the three query heads within each group can be arranged arbitrarily without affecting functionality, *e.g.*, applying $(4, 6, 5, 2, 3, 1)$ to the six query heads. To put it more formally, the permutation σ applied on query heads (not query weights) is similarly induced by π and h_k permutations for each group:

$$\sigma(g \cdot i - j) = g \cdot \pi(i) - \tau_i(g - j) + 1 \text{ for } 1 \le i \le h_k, 0 \le j < g, \quad (14)$$

where τ_i is a permutation of g elements that rearranges the query heads within the i-th group. We denote this induced permutation σ as $\mathrm{BP}(\pi; \tau_1, \cdots, \tau_{h_k})$, which differs from Eq. (12) in that elements in each block are permuted according to a given permutation τ_i.

[2]If E and W_{lm} share weights, the permutation needs to be applied only once.

The query, key, value, and projection weights are permuted by:

$$\begin{aligned} W_q{}' &= (W_q)_{:,\mathrm{BP}(\sigma;d_k)}, \quad W_k{}' = (W_k)_{:,\mathrm{BP}(\pi;d_k)}, \\ W_v{}' &= (W_v)_{:,\mathrm{BP}(\pi;d_k)}, \quad W_o{}' = (W_o)_{\mathrm{BP}(\sigma;d_k),:}. \end{aligned} \quad (15)$$

Since the construction of σ depends on both π and τ_i, the dimension permutation operation for GQA takes not only π but also $\tau_1, \cdots, \tau_{h_k}$ as input, and we have $R_{\text{GQA}} = \Pi_{h_k} \times (\Pi_g)^{h_k}$. Therefore, the total number of permissible permutations $|R_{\text{GQA}}|$ for grouped query attention is $h_k! g!^{h_k}$.

5.1.4 Hidden Dimension. $\textsc{Perm}(\pi, l^{(i)}, T_{\text{hid}})$ permutes the hidden dimension at the i-th key head at the l-th self-attention layer, *i.e.*, $(W_q^{(i)})' = (W_q^{(i)})_{:,\pi}$ and $(W_k^{(i)})' = (W_k^{(i)})_{:,\pi}$. When GQA is employed, the permutation $\pi \in R_{\text{hid}} = \Pi_{d_k}$ needs to be applied to the entire query group paired with the key head. Additionally, the sequence of rotation frequencies θ_i needs to be permuted according to π to maintain model invariance if rotary embeddings are used. However, θ_i are not part of the model weights, and they are often shared across all heads. Therefore, hidden dimension permutation is only meaningful to models using absolute positional embeddings.

5.2 Permutation Extraction

To retrieve permutations from a watermarked model f', this method compares the weights of f (original model) and f' at each slot s_i to extract the permutation π_i (and applies the inverse operation $\textsc{Perm}(\pi_i^{-1}, l_i, t_i)$ if t_i is T_{emb}). Different from scaling/unscaling (SU) and invertible matrix in query-key products (IM) transformations that take continuous parameters (e.g., scaling vectors), weight permutation (WP) uses discrete parameters, i.e., permutations. While the discreteness of permutations offers greater robustness when extracting the transformation parameters, as recovering the permutation can be less susceptible to numerical noise, it also makes the extraction process computationally more expensive.

To address this challenge, we propose an efficient permutation extraction algorithm based on linear assignment solving. Given a weight matrix $W' \in \mathbb{R}^{s \times d}$ from a watermarked model, we assume that it can be represented in the following form:

$$W' = WC_\pi + \epsilon \quad (16)$$

where $C_\pi \in \mathbb{R}^{d \times d}$ is the column permutation matrix determined by $\pi \in \Pi_d$ that we want to extract, *i.e.*, $(C_\pi)_{i,j} = 1$ when $i = \pi(j)$ and 0 otherwise, $W \in \mathbb{R}^{s \times d}$ is the original weight matrix accessible to the model provider, and $\epsilon \in \mathbb{R}^{s \times d}$ is the noise introduced during model deployment or modification. Given W and W', the goal is to recover the permutation π. We frame the problem as a matching problem between the columns of W and W'. Each column of W is paired with a column of W', with the matching cost defined by the Euclidean distance between them. The objective is to recover π by minimizing the total cost, which can be formulated as a linear assignment problem. The following theorem shows that under a mild assumption on the noise level, the solution to this matching problem is unique and the desired permutation π can be obtained (complete proof in Appendix A).

Theorem 5.1. *If a permutation π satisfies Eq.* (16) *and the following condition holds:*

$$\min_{i \ne j} \|W_{:,i} - W_{:,j}\|_2 > 2 \max_i \|\epsilon_{:,i}\|_2, \quad (17)$$

then π is the unique optimal solution to the following linear assignment problem:

$$\pi = \arg\min_{\sigma \in \Pi_d} \sum_{i=1}^{d} cost(i, \sigma(i)) = \arg\min_{\sigma \in \Pi_d} \sum_{i=1}^{d} \|W_{:,i}{}' - W_{:,\sigma(i)}\|_2, \quad (18)$$

where $cost(i, \sigma(i))$ is the cost of matching the i-th column of W with the $\sigma(i)$-th column of W'.

The condition Eq. (17) ensures that the noise level is sufficiently small relative to the minimum column distance of W. This condition is practical and often satisfied in real-world scenarios, as the noise introduced during model deployment or modification (e.g., quantization) typically preserves the distinctiveness of the columns.

To extract permutations applied on rows of a weight matrix, simply transpose the matrix to convert it to a column permutation extraction problem. For block permutations, we can use another cost function $\text{cost}_{\text{blk}}(i, \sigma(i))$ that considers the block-wise difference:

$$\text{cost}_{\text{blk}}(i, \sigma(i)) = \sum_{j=1}^{b} \|W_{:,i \cdot b+j}{}' - W_{:,\sigma(i) \cdot b+j}\|_2, \quad (19)$$

where b is the size of each block (*e.g.*, hidden dimension d_k). We can also prove the following theorem for block permutation extraction (complete proof in Appendix A).

Theorem 5.2. *Under the assumption of Eq.* (17)*, if the permutation that satisfies Eq.* (16) *is a block permutation $\pi^b \in \Pi_d^b$, then π is the unique optimal solution to the following linear assignment problem:*

$$\pi = \arg\min_{\sigma \in \Pi_{d/b}} \sum_{i=1}^{d/b} cost_{blk}(i, \sigma(i)). \quad (20)$$

The time complexity of the proposed extraction algorithm is $O(d^3 + sd^2)$ using Jonker-Volgenant algorithm [29] for linear assignment, and it does not require additional storage space. As a comparison, the permutation extraction method employed in the previous study [18] essentially performs a brute-force search over all permutations used for watermarking, which has a theoretical time complexity of $O(Dsd)$ where D is the total number of inserted permutations by the provider. It also requires $O(Dd)$ storage space for all inserted permutations. Therefore, the cost of the brute-force extraction method grows linearly with the number of inserted permutations. A speed-up in the order of 100× can be achieved by only considering a subset of columns (rows) for extraction, but this is an approximation method that can result in up to 30% error rate in the extracted permutations. In contrast, our linear assignment-based extraction method not only guarantees a unique optimal solution but also supports block permutation extraction. This efficiency gain, coupled with the guarantee of optimality, underscores the effectiveness of our proposed method over existing approaches.

5.3 Considerations for Potential Attacks

The weight permutation watermarking method without ECCs faces two adaptive attacks, including watermark obfuscation and watermark removal and forgery, which are discussed in the following.

Watermark obfuscation attacks can be launched by applying the scaling/unscaling (SU) or invertible matrix (IM) transformations to the watermarked model. These transformations disrupt the permutation extraction process by violating the permutation assumption Eq. (16) for the query and key weight matrices. More specifically, the SU transformation scales the parameters in layer normalization (and its variants) by a positive constant vector $\alpha \in \mathbb{R}^d$, and rescales the rows of the following linear layer by the same vector, *i.e.*,

$$\text{LN}' = \alpha \odot \text{LN}, \quad W_1' = \text{diag}(1/\alpha) W_1, \quad (21)$$

where $\odot$ denotes element-wise multiplication, $1/\alpha$ is the element-wise reciprocal of α, and $\text{diag}(\cdot)$ constructs a diagonal matrix. The IM transformation multiplies the query weight matrix W_q by an invertible matrix $Q \in \mathbb{R}^{hd_k \times hd_k}$ and the key weight matrix W_k by its inverse Q^{-1} to preserve Eq. (1), *i.e.*,

$$W_q' = W_q Q, \quad W_k' = W_k \left(Q^{-1}\right)^\top. \quad (22)$$

If rotary embeddings are used, then Q must be a block diagonal matrix with 2×2 sub-matrices on the diagonal (proof in Appendix A), where each sub-matrix is a rotation matrix multiplied by a scaling factor.

Watermark removal and forgery attacks are essentially caused by the lack of a mechanism to reliably recover the original model identifiers from watermarks with corrupted permutations. The adversary could apply a random permutation σ to a permutation slot, resulting in a permutation corruption $\pi' = \sigma \circ \pi$ during watermark extraction. The more permutations are corrupted, the more likely the extracted watermark cannot be matched with any of the inserted watermarks. Even worse, the extracted watermark may be matched with a completely different watermark, resulting in a forged watermark. A simple approach to match the extracted watermark with the inserted watermarks is to use the nearest neighbor search with some predefined threshold value [18], where the extracted watermark is considered valid if the distance to the nearest inserted watermark is below the threshold. However, this approach cannot reliably determine the model identifier because of the following two reasons:

(1) The closest watermark to the extracted sequence of permutations may not be unique if the inserted permutations are not carefully designed.
(2) The threshold value for the nearest neighbor search must be chosen empirically to balance between sensitivity and specificity, and it cannot provide a strict guarantee of accuracy.

Therefore, the nearest neighbor search matching method could not serve as a reliable watermark identification mechanism to defend against permutation corruption, allowing adversaries to potentially remove or forge watermarks.

6 Error Correction Enhanced Weight Permutation

To defend against watermark removal and forgery attacks, we incorporate error correction codes and a permutation mapping into the weight permutation watermarking method. The intuition is that when extracted permutations are corrupted by adversaries, the error correction codes can help recover the original model identifier, thus enhancing the robustness of the watermarking scheme. In addition, the encoding process can be viewed as a lightweight encryption method, which can prevent adversaries from directly reading the

Algorithm 1: Robust weight permutation watermarking

Data: Transformer model f, watermarked model f', model identifier id, generalized Reed-Solomon code $\mathrm{GRS}_{n,k}$

function *insert-model-identifier(f, id,* $\mathrm{GRS}_{n,k}$*)* **is**
  $c \leftarrow \mathrm{GRS}_{\mathrm{ENC}}(id)$;
  $f' \leftarrow$ *insert-permutations*$(f, \phi(c))$;
  return f';

function *extract-model-identifier(f, f',* $\mathrm{GRS}_{n,k}$*)* **is**
  $\sigma \leftarrow$ *extract-permutations*(f, f');
  $id \leftarrow \mathrm{GRS}_{\mathrm{DEC}}(\phi^{-1}(\sigma))$;
  if *id is undefined* **then return** FAIL;
  return id;

model identifier from the watermarked model. To defend against watermark obfuscation attacks, we propose an improvement for the linear assignment-based permutation extraction method. For clarity, we use the term *watermark* to refer to the sequence of permutations applied to the model, and *model identifier* to refer to the actual message that identifies the model.

The overview of the error correction enhanced robust weight permutation watermarking method is shown in Algorithm 1. The model identifier id is given as a sequence of k symbols in a finite field GF of size q. Then it is encoded into a codeword $c = [c_1, \cdots, c_n]$ of n symbols using a generalized Reed-Solomon code $\mathrm{GRS}_{n,k}(\boldsymbol{e}, \boldsymbol{v})$. With a permutation mapping ϕ, the codeword c is interpreted as a sequence of permutations $[\pi_1, \cdots, \pi_n]$, which is inserted into the model as a watermark. After the adversary applies adaptive attacks, the provider extracts the permutations $[\sigma_1, \cdots, \sigma_n]$ and decodes it using ϕ^{-1} and $\mathrm{GRS}_{\mathrm{DEC}}$ to recover the original model identifier. As long as the number of corrupted permutations introduced by adversaries does not exceed the correction capability of the error correction code, the original model identification identifier can be uniquely recovered. Additionally, the corrupted permutations can be detected during permutation extraction with a very high probability. The stealthiness of the code can be further enhanced by choosing the parameters $\boldsymbol{e}$ and $\boldsymbol{v}$ carefully.

6.1 Permutation Mapping

The permutation mapping ϕ is a parameterized injective function that maps symbols in the finite field GF to a subset of permutations of d elements, *i.e.*, $\phi_d : \mathrm{GF} \to \Pi_d$. Its inverse ${\phi_d}^{-1}$ is a partial function that converts a subset of Π_d back to the symbols in GF, *i.e.*, ${\phi_d}^{-1} : \Pi_d \rightharpoonup \mathrm{GF}$. This subset should be chosen carefully because some permutations may not be safe for watermarking. For example, if the permutation has many fixed points (*i.e.*, $\pi(i) = i$) and the adversaries have access to parts of multiple different models, they may compare the arrangement of weights in different models to find common fixed points, thus potentially revealing part of the original private model. Therefore, we *restrict the set of permissible permutations R_i in each slot to permutations with no fixed points*, also called *derangements*, denoted by $\mathcal{D}_d$ where d is the number of elements. The size of $\mathcal{D}_d$ is given by the subfactorial $!d$, which is the nearest integer to $d!/e$.

Suppose we have a ranking function r_d for all derangements of d elements [42], *i.e.*, $r_d : \mathcal{D}_d \to \{1, \cdots, !d\}$ is a bijective mapping. The inverse ${r_d}^{-1}$ is denoted as the unranking function. We can then construct ϕ_d as follows:

$$\phi_d(i_{\mathrm{GF}}) = {r_d}^{-1}\left(\alpha \cdot \mathrm{integer}(i_{\mathrm{GF}})\right), \tag{23}$$

where the i-th symbol i_{GF} in GF is mapped to the $(\alpha \cdot i)$-th derangement, and the scaling factor $\alpha = \lfloor !d/q \rfloor$ obfuscates the mapping and ensures that the derangements are uniformly distributed. For an extracted permutation $\pi \in \Pi_d$, it is decoded as:

$${\phi_d}^{-1}(\pi) = \begin{cases} \left(\frac{r_d(\pi)}{\alpha}\right)_{\mathrm{GF}}, & \text{if } \pi \in \mathcal{D}_d \text{ and } \alpha \mid r_d(\pi), \\ \uparrow, & \text{otherwise.} \end{cases} \tag{24}$$

In other words, if π is not a derangement, or $r_d(\pi)$ is not a multiple of α, then π does not belong to the chosen subset of derangements for watermarking, in which case we denote ${\phi_d}^{-1}(\pi) \uparrow$ as undefined. It is easy to see that ${\phi_d}^{-1}(\phi_d(s_{\mathrm{GF}})) = s_{\mathrm{GF}}$ for all $s_{\mathrm{GF}} \in \mathrm{GF}$.

For GQA, we need to permute the groups and the individual query heads within each group. Consequently, we need a sophisticated permutation mapping $\phi_{h_k,g}$ that can encode a symbol in GF to a derangement of order h_k and h_k derangements of order g where h_k is the number of key heads and g is the group size. We first construct a pair of coprime integers $u_1 < (!g)^{h_k}$ and $u_2 < !h_k$ such that u_1 and u_2 are the largest primes less than $(!g)^{h_k}$ and $!h_k$. Then we obtain two remainder values m_1 and m_2 by:

$$\begin{cases} \tilde{s} = \beta \cdot \mathrm{integer}(s_{\mathrm{GF}}) \equiv m_1 \mod u_1, \\ \tilde{s} = \beta \cdot \mathrm{integer}(s_{\mathrm{GF}}) \equiv m_2 \mod u_2, \end{cases} \tag{25}$$

where $\beta = \lfloor \frac{u_1 u_2}{q} \rfloor$ scales s to the range of $[0, u_1 u_2)$. Finally, we define $\phi_{h_k,g}$ as follows:

$$\phi_{h_k,g}(s_{\mathrm{GF}}) = \left(\phi_{h_k}(m_2), \phi_g(m_{1,1}), \cdots, \phi_g(m_{1,h_k})\right), \tag{26}$$

where $(m_{1,1}, \cdots, m_{1,h_k})$ are the base-$!g$ representation of m_1. The inverse ${\phi_{h_k,g}}^{-1}(\sigma, \pi_1, \cdots, \pi_{h_k})$ is defined similarly as ${\phi_d}^{-1}$: use ${\phi_{h_k}}^{-1}$ and ${\phi_g}^{-1}$ to decode σ and π_i back to remainders m_1 and m_2, and then solve Eq. (25) by the Chinese remainder theorem to obtain the symbol s_{GF}. Since u_1 and u_2 are coprime, the Chinese remainder theorem guarantees that a unique solution $\tilde{s}$ exists in the range of $[0, u_1 u_2)$. If ${\phi_{h_k}}^{-1}(\sigma)$ or any ${\phi_g}^{-1}(\pi_i)$ is undefined, or β does not divide the solution $\tilde{s}$, then the inverse is considered as undefined. Similarly, we have ${\phi_{h_k,g}}^{-1}(\phi_{h_k,g}(s_{\mathrm{GF}})) = s_{\mathrm{GF}}$ for all $s_{\mathrm{GF}} \in \mathrm{GF}$.

By lifting the definitions of ϕ_d and $\phi_{h_k,g}$ to sequences of symbols, we obtain the overall permutation mapping ϕ. The design of ϕ makes it easy to detect corrupted permutations caused by adversaries. Given a derangement π that is inserted into model f', assume that the adversary corrupts it to $\pi' = \sigma \circ \pi$, then this corruption can be detected by the provider if $\phi^{-1}(\pi')$ is undefined, whose probability can be bounded by the following theorem (complete proof provided in Appendix B).

THEOREM 6.1. *Suppose the adversary corrupts a derangement $\pi \in \mathcal{D}_d$ by compositing it with a permutation $\sigma \in \Pi_d$, resulting in the extracted permutation $\pi' = \sigma \circ \pi$. Denote the probability that $\phi^{-1}(\pi')$ is defined as $p_{corrupt}$,* i.e., *the corruption is not detected during decoding, then $p_{corrupt} \le \frac{q}{d!}$ if $\sigma \sim \mathcal{U}(\Pi_d)$.*

Algorithm 2: Reed-Solomon code parameters setting

Input: Total number of model identifiers N, maximal probability for failing to detect a corruption m_p, codeword length n
Output: Finite field size q, identifier length k

$u \leftarrow \lfloor \log_2(m_p \min_i |R_i|) \rfloor$;
for $k = 1$ **to** $n - 1$ **do**
  $l \leftarrow \lceil \log_2(N^{\frac{1}{k}}) \rceil$;
  if $l \leq u$ **then**
    $q \leftarrow 2^l$;
    return q, k;

6.2 Error Correction Code Setting

We use generalized Reed-Solomon codes because they (1) operate over a finite field instead of only binary symbols, which is suitable for the large permutation space, and (2) have a high correction capability. To setup a generalized Reed-Solomon code, we need to specify four parameters: codeword length n, identifier (message) length k, finite field size q, and the other parameters $\boldsymbol{e}$ and $\boldsymbol{v}$.

We set the codeword length n to the maximum number of permutation slots in the model to maximize the tolerance to corrupted permutations. The parameters $\boldsymbol{e}$ and $\boldsymbol{v}$ can be arbitrarily chosen by the provider to stay stealthy.

The settings of q and k depend on the robustness requirements and there is a trade-off between them. On the one hand, the error correction capability of the code is determined by $n-k$, so we would like a smaller k for better correction capability. On the other hand, a smaller q is also preferred to lower the chance that the provider fails to detect a corrupted permutation, as shown in Theorem 6.1. Being able to detect corrupted permutations contributes to the overall robustness, because when the corruption is detected, it is essentially an erasure with known position, while if the corruption is not detected, it becomes an error that is more expensive to correct. Therefore, the smaller the value of q, the more likely the provider can detect corrupted permutations, and the more corruption can be corrected by the error correction code. However, we cannot have a small k and a small q at the same time, because the total number of different model identifiers is q^k, which should be sufficiently large for distributing models.

To balance the trade-off, we first set a total number of model identifiers N that the provider needs to manage and a maximal probability m_p that the provider can tolerate for failing to detect a corrupted permutation. Then we determine q and k by maximizing the error correction capability while satisfying the constraints:

$$\begin{aligned} &\max_{q=2^t,k} \quad n-k \\ &\text{s.t.} \quad q^k \geq N, \quad \frac{q}{\min_i |R_i|} \leq m_p, \quad t \in \mathbb{N} \end{aligned} \tag{27}$$

where $\min_i |R_i|$ is the minimal number of permissible derangements across all slots. The second constraint is a very conservative estimation of m_p because $|R_i|$ is usually much smaller than $d!$ (or $h_k! g!^{h_k}$ for GQA slots). We've put the provider in a very unfavorable position by assuming that the adversary's attack always makes the extracted permutation fall into the set of permissible derangements. Such a defensive strategy ensures a high robustness of the watermarking method. We additionally require q to be a power of two for efficient implementation of the finite field arithmetic. Algorithm 2 presents the procedure to determine q and k. For example, if we set $m_p = 0.0001$ and $N = 10^7$ for Llama3.2-1B, then $q = 2^{24}$ and $k = 1$, which guarantees that the provider can detect corrupted permutations with a probability more than 99.99% and manage up to 10^7 different model identifiers. At the same time, the error correction capability of the code is maximized to $n - k = 32$.

6.3 Robust Permutation Extraction

With slight modifications, the proposed linear assignment-based extraction method can defend against scaling/unscaling attacks (*i.e.*, SU transformation), which violate the assumption in Eq. (16) by multiplying the rows with a constant vector α:

$$W' = \text{diag}(\alpha) W C_\pi + \epsilon. \tag{28}$$

To defend against this attack, we first normalize the rows of W and W' to cancel out the scaling/unscaling effect, and then solve Eq. (18) to extract the permutation. The uniqueness of the optimal solution can still be guaranteed under the accordingly adjusted assumption about the scale of ϵ.

Theorem 6.2. *Solution of Eq.* (28) *is also the unique optimal solution to the linear assignment problem in Eq.* (18) *after normalizing the rows of W and W', if the normalized matrices satisfy the condition in Eq.* (16)*, where the normalization is defined as:*

$$\text{normalize}(W) = \text{diag}\left(\frac{1}{\|W_{1,:}\|_2}, \cdots, \frac{1}{\|W_{s,:}\|_2} \right) W. \tag{29}$$

The invertible matrices in QK products (*i.e.*, IM transformation) can be defended more straightforwardly. We extract permutations from value weights (*i.e.*, W_v), instead of query and key weights (*i.e.*, W_q and W_k). Because the value weight is not affected by IM transformation, the permutation extracted from W_v can be directly used to decode the model identifier.

6.4 Summary

Compared with the vanilla FIT-based watermarking method, the proposed error correction enhanced weight permutation watermarking approach offers the following advantages:

(1) It can defend against adaptive attacks based on SU and IM transformations, and we've established a strict guarantee for this property through Theorem 6.2.
(2) It can tolerate a certain level of permutation corruption, and the provider can detect corrupted permutations with a high probability, thus correcting up to $n - k$ erasures, which makes the watermark removal and forgery attacks more difficult.
(3) It supports the commonly used GQA mechanism in large language models, and has a lower time complexity for watermark extraction compared to the brute force search method.
(4) The parameter m_p has a direct relationship with the robustness level and can be set according to the provider's requirements, whereas the threshold value in the vanilla FIT-based method needs to be empirically determined.

The error correction code essentially serves as a matching mechanism between model identifiers and extracted permutations. The matching result is guaranteed to be unique and correct within the error correction capability, as the generalized Reed-Solomon code is

Table 1: Watermark settings.

Model	n	q	k	#Blk	d	h	d_{ff}
Llama3.2-1B	33	2^{24}	1	16	2,048	8×4	8,192
Llama3.2-3B	57	2^{8}	3	28	3,072	8×3	8,192
Llama2-7B	65	2^{24}	1	32	4,096	32	11,008
Llama3.1-8B	65	2^{24}	1	32	4,096	8×4	14,336
Gemma-7B	57	2^{24}	1	28	3,072	16	24,576
Ministral-8B-I	73	2^{24}	1	36	4,096	8×4	12,288

Table 2: Permutation sampling strategies by the adversary.

Strategies	Description	Example
Uniform	Sample a permutation from Π_d with equal probability (excluding identity).	$(2, 4, 3, 1)$
Swap	Sample a permutation by swapping two elements uniformly at random.	$(2, 1, 3, 4)$
Cycle	Sample a permutation by generating a random cycle.	$(2, 3, 4, 1)$
Derangement	Uniformaly sample a derangement from P_d at random.	$(2, 4, 1, 3)$

Table 3: Model utility evaluation results for baseline method.

Model	PPL(w/o)	Setting	PPL changes	Distortion
Llama2-7B	9.937	WP	0.0	0.16%
		SU	+0.001	0.17%
		IM	0.0	0.16%
		All	+0.001	0.17%
Gemma-7B	8.934	WP	+0.002	7.67%
		SU	–	–
		IM	0.0	7.92%
		WP+IM	+0.002	7.91%

Table 4: Model utility evaluation results for proposed method.

Model	PPL (w/o)	PPL (w/)	PPL changes	Distortion
Llama3.2-1B	9.194	9.195	+0.001	1.45%
Llama3.2-3B	7.449	7.449	0.0	1.15%
Llama2-7B	9.937	9.937	0.0	0.13%
Llama3.1-8B	6.151	6.151	0.0	1.17%
Gemma-7B	8.934	8.935	+0.001	7.68%
Ministral-8B-I	13.102	13.102	0.0	1.09%

a maximum distance separable (MDS) code. In contrast, the vanilla FIT-based method uses a nearest neighbor search method to match the extracted permutation with the model identifier, which not only has a higher computational overhead but also lacks a guarantee of uniqueness and correctness.

7 Evaluation

In this section, we first evaluate the utility of watermarked models. Then we verify the theoretical guarantee on the probability of detecting corruption in the watermark by simulating various attack settings. We further evaluate the robustness and efficiency of the proposed watermarking method.

7.1 Experiment Setting

Model. We use six language models with varying sizes for evaluation, including Llama3.2-1B, Llama3.2-3B, Llama2-7B, Llama3.1-8B, Gemma-7B, and Ministral-8B-Instruct-2410. These models all use pre-normalization with RMSnorm [5], rotary position embeddings [59], and SwiGLU activation [55], except for Gemma-7B, which uses tanh-based approximation of GELU [26]. They also employ GQA except for Llama2-7B and Gemma-7B.

Watermark settings. We set m_p to 0.0001 and N to 10^7 for all watermarking experiments, and the parameters $\boldsymbol{e}$ and $\boldsymbol{v}$ of $\text{GRS}_{n,k}$ are randomly generated. The ranking and unranking functions for derangements are implemented using algorithms in [42], both with linear time complexity. Identifier length k, finite field size q, codeword length n, and dimensions are shown in Table 1, where #Blk is the number of attention blocks, h is the number of heads in the multi-head attention (in the form of $h_k \times g$ if GQA is used), and the meanings of other dimensions are consistent with Section 2.

Attacks and model modifications. The probability of failing to detect corruption given in Theorem 6.1 is derived assuming a uniform sampling strategy for the adversary. In practice, different strategies can be employed by the adversary to maximize the probability of undetected corruption. However, synthesizing the optimal adversary strategy is a challenging problem, as it highly depends on the underlying permutation mapping that is assumed to be unknown to the adversary (ranking and unranking functions can be chosen arbitrarily). Therefore, we consider three additional strategies when estimating the probability: swap, cycle, and derangement, as explained in Table 2. These strategies are also used to evaluate the robustness against watermark forgery attacks. We evaluate the robustness against commonly used model modification techniques, including quantization, pruning, and fine-tuning.

Baselines. We compare the proposed method with the baseline method described in [18], which utilizes function invariant transformations, including weight permutation (WP), scaling/unscaling (SU), and invertible matrices in QK products (IM). The baseline method applies one or a combination of these transformations to watermark the models. Since it does not support models with GQA, we exclude GQA models from the baseline evaluation.

7.2 Model Utility

7.2.1 Utility Metrics. Although weight permutations theoretically preserve the model's functionality, floating-point errors can lead to observable utility distortions, particularly when the models are loaded in 16-bit precision to optimize memory usage and inference speed. We measure the perplexity of models on the WikiText-2 dataset [41] before and after watermarking to evaluate the generation quality changes, denoted as PPL (w/o) and PPL (w/), respectively. The input token sequence length is set to 4,096 for calculating the perplexity, which measures how well the model predicts against the ground truth. The lower the perplexity, the better the

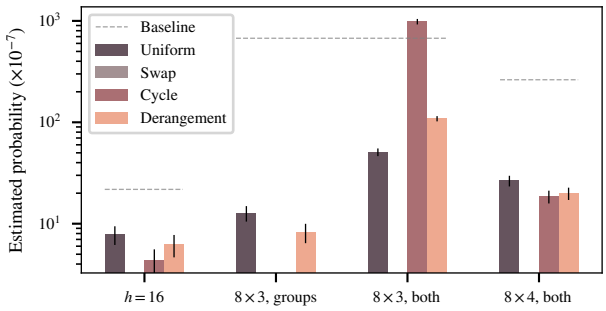

Figure 2: Estimated probability of failing to detect corruption under different parameter settings and sampling strategies.

model performs. However, the focus of this evaluation is the relative change in perplexity after watermarking. We also calculate the token distortion, which is the percentage of top-1 tokens given by the watermarked model that are different from the original model's predictions.

7.2.2 Results. We report the perplexity changes and token distortion for the baseline method and our proposed method in Table 3 and Table 4, respectively. The perplexity changes are almost negligible for all models, with the largest change being 0.002 for Gemma-7B when applying the WP transformation. The token distortion is around 1% for most models, except for Gemma-7B, where it reaches 7%. SU transformation does not maintain invariance for Gemma-7B because it adopts a non-linear scaling in layer normalization. Therefore, we only report the results for WP, IM, and their combination. These results indicate that the generation quality remains largely unaffected by the watermarking process, thus preserving the utility of the original models.

7.3 Probability of Failing to Detect Corruption

7.3.1 Estimation Method. We investigate the probability of failing to detect corruption through simulations under different sampling strategies by the adversary. We simulate corruption using the configurations in Table 1, targeting T_{heads} slots of the models. This is because h is smaller than d and d_{ff}, making the attention head permutation slots more vulnerable to undetected corruption. For GQA, we consider three corruption settings: (1) permuting only the groups, (2) permuting only the heads within one group, and (3) permuting both groups and all heads within each group. Each simulation is repeated 100,000,000 times, and the estimated probability is calculated as the ratio of undetected corruptions to the total number of simulations.

7.3.2 Results. In Fig. 2, we report the estimated probability under different attack settings and their 95% confidence intervals. Additionally, we show the baseline probabilities considered when setting the parameters of the Reed-Solomon codes in Eq. (27), represented by grey dashed lines. Results for $h = 32$, 8×4 with only group permutation, and all settings with only head permutation within one group are not shown because the estimated probabilities are all 0. The estimation results show that cycle and derangement strategies are more likely to introduce undetected corruption than the uniform strategy. Swap strategy fails across all settings, and just permuting heads within one group for GQA is also ineffective in evading detection. For GQA, permuting both groups and all heads within each group is the most effective setting, and the estimated probability when using a cycle strategy sometimes even exceeds the theoretical baseline value. Despite that, all estimation results are below the predefined threshold $m_p = 0.0001$, indicating that the proposed watermarking method can effectively detect corruption under different attack settings.

7.4 Robustness

We evaluate the robustness of the proposed method against (1) various model modifications, (2) watermark obfuscation attacks, including the SU and IM transformations, and (3) watermark removal and forgery through compositing random permutations.

7.4.1 Model Modifications. After the distribution of watermarked models, users may modify the models in various ways for downstream tasks, and adversaries may use these modifications to remove the watermarks. We evaluate the robustness against three common model modification techniques for transformer models: quantization, pruning, and fine-tuning. Firstly, we provide a brief description of the settings for these techniques in our experiments. Full details of the settings are provided in Appendix C.

Quantization is a common technique to reduce the memory footprint and improve inference speed by converting parts of the model to lower precision. Post-training quantization methods are among the most popular model compression techniques, which quantizes the weights after training. We include four post-training quantization settings in the experiments: 8-bit static quantization method in PyTorch, 4-bit, 3-bit, and 2-bit quantization by AutoGPTQ [21].

Pruning is widely used to reduce the model size for deployment. Unstructured pruning sets the weights that contribute less to the model's performance to zero, while structured pruning removes entire neurons, channels, or even layers. We include two state-of-the-art unstructured pruning methods, SparseGPT [20] and Wanda [60], with pruning ratios of 0.5 and 0.7.

Fine-tuning is broadly used to adapt pre-trained models to specific downstream tasks, and it is also a common strategy to remove watermarks. Extensive fine-tuning may lead to large deviation in the model weights, thus potentially affecting watermark extraction. We use LoRA [27] to fine-tune watermarked models[3] for different data volumes ranging from 1 million to 100 million tokens on the WikiText-2 dataset [41]. This dataset is also used for calibration in quantization and pruning experiments.

Table 5 shows the number of symbols (permutations) that are different from the original codewords in the extracted watermarks after applying these model modifications, where S-0.5 (resp. W-0.7) denotes the SparseGPT (resp. Wanda) pruning method with a pruning ratio of 0.5 (resp. 0.7). Most of the watermarks can be extracted correctly with no corrupted permutations, even after 2-bit quantization, 70% pruning, and 100M token fine-tuning. The only exception is the 100M fine-tuning on Llama3.1-8B, where one permutation is corrupted. Since the number of corruptions is within the tolerance, the model identifier can still be extracted correctly. We've demonstrated that corruption in the watermarks can be detected with

[3] LoRA weights are merged into the watermarked model.

Table 5: Robustness evaluation results on model modifications.

Model	Quantization				Pruning				Fine-tuning				
	8-bit	4-bit	3-bit	2-bit	S-0.5	S-0.7	W-0.5	W-0.7	1M	5M	10M	50M	100M
Llama3.2-1B	0	0	0	0	0	0	0	0	0	0	0	0	1
Llama3.2-3B	0	0	0	0	0	0	0	0	0	0	0	0	0
Llama2-7B	0	0	0	0	0	0	0	0	0	0	0	0	0
Gemma-7B	0	0	0	0	0	0	0	0	0	0	0	0	0
Llama3.1-8B	0	0	0	0	0	0	0	0	0	0	0	0	0
Ministral-8B-I	0	0	0	0	0	0	0	0	0	0	0	0	0

high probability in Section 7.3, and this corruption in Llama3.1-8B is also detected by the watermark extraction process. Notably, the extensively fine-tuned models (100M tokens) exhibit severe performance degradation, but the watermarks remain recoverable. This is because the structure of the embedded permutations is preserved despite the extensive modifications. The results indicate that the proposed watermarking method is robust against common model modifications. This also serves as an empirical validation of the theoretical guarantee presented in Theorem 6.2.

7.4.2 Watermark Obfuscation via Functional Invariant Transformations. As discussed in Section 5, the scaling/unscaling (SU) transformation and the invertible matrices in QK products (IM) transformation can preserve the functionality of the model while violating our assumption Eq. (16) for permutation extraction. We proposed an improvement for the extraction method to defend against these transformations, and we evaluate the robustness of the proposed method against these detection evasion attacks. After inserting watermarks, we apply both SU and IM transformations to all layers of the watermarked models, and then extract permutations and compare them with the ground truth. Parameters for these transformations are set to the same values as in [18]:

(1) Scaling/unscaling (SU): scaling factor α_i satisfies $\log_{10}(\alpha_i) \sim \mathcal{U}(-1, 1)$ and the transformation is applied to all layer normalization layers and their following linear layers.
(2) Invertible matrices in QK products (IM): rotation degrees are sampled from $\mathcal{U}(0, 2\pi)$ and the transformation is applied to all query and key weights in the attention layers.

We repeated the attacks 10 times on each permutation slot in each model, and *all permutations are extracted correctly with no corruption*, as guaranteed by Theorem 6.2. The results demonstrate that the proposed method can effectively defend against watermark obfuscation attacks based on functional invariant transformations.

7.4.3 Watermark Removal and Forgery. An adversary aware of the watermarking scheme may attempt adaptive attacks to remove or forge the watermarks. For the baseline method, its authors noted that the adversary can apply a series of FITs to the watermarked models to make the extracted watermark invalid, thus removing the watermark. In fact, the extracted parameters may even be decoded into another valid model identifier, constituting a forgery. The success of such adaptive attacks highly depends on the properties (e.g., minimum distance) of the set of possible identifiers – a critical aspect not thoroughly addressed in the baseline work [18]. For instance, using simple integer sequences as identifiers, as in the baseline, means two identifiers differing by only one element could potentially be swapped via a single FIT operation, resulting in a forgery. Designing robust identifier sets resilient to such manipulations is non-trivial, underscoring the limitations of the baseline. Therefore, our evaluation focuses on the robustness of our method against removal and forgery, rather than re-validating the baseline's known weakness.

For the proposed watermarking method, removal and forgery are more challenging due to the integration of error correction codes with permutation mapping. An adversary can apply permutations to corrupt the watermark, but the original model identifier can still be recovered as long as the number of corrupted permutations is within the GRS code's correction capability. To compromise the watermark, an adversary must introduce corruptions exceeding this capability (specifically, more than $n - k$). In this case, two possible outcomes may occur: (1) Removal: the extracted permutations cannot be decoded into a valid model identifier; (2) Forgery: the extracted permutations happen to decode into another valid model identifier. The outcome depends on the specific errors introduced and the structure of the GRS code, which is beyond the adversary's control.

We evaluate our method's robustness against watermark removal and forgery attacks following the most effective setting identified in Section 7.3. For each model, we consider two scenarios: (1) removal attacks applying $n - k$ random permutations to randomly selected layers, and (2) forgery attacks applying n random permutations to all layers. Each attack is repeated 2,000,000 times independently for watermarked Llama3.2-1B and Llama3.2-3B.

All removal attack attempts failed and the corruptions were all corrected by the GRS code. For forgery attacks with n corruptions, we observed *no successful forgery attempts* for both models, and the extracted permutations cannot be decoded into any valid model identifier. This result indicates that a determined adversary can remove the watermarks by applying a large number of permutations, but the probability of forging a new watermark appears extremely low. During the extraction process, all corrupted permutations were detected. The observed frequency of undetected corruption is far below the estimated probability in Section 7.3, which is expected because we were considering the worst-case scenario in the estimation. More experiment details are provided in Appendix D.

7.5 Efficiency

Large language models have vast numbers of parameters, which makes traditional watermarking methods that require training computationally expensive. The proposed method is much more efficient as it is training-free. The insertion only requires permuting the weights of the model, thus it can be done efficiently on the CPU.

Table 6: Average time cost of the proposed method for different model sizes (in seconds).

Model	n	Insertion (s)	Extraction (s)
Llama3.2-1B	33	1.4	6.7
Llama3.2-3B	57	2.0	18.4
Llama2-7B	65	4.0	33.7
Llama3.1-8B	65	4.8	52.7
Gemma-7B	57	7.6	228.0*
Ministral-8B-I	73	4.5	45.7
Stable LM-2-12B	81	7.4	124.3
Llama2-13B	85	16.1	129.2

*Extraction falls back to sequential mode due to memory constraints.

Table 7: Average time cost of function invariant transformations for different model sizes (in seconds).

Model	Insertion (s)			Extraction (s)		
	Perm.	Scaling	QK	Perm.	Scaling	QK
Llama2-7B	4.6	1.0	1.0	423.1	15.5	4.7
Gemma-7B	5.6	1.2	0.4	705.3	19.1	3.3
Llama2-13B	9.1	2.8	2.1	828.1	34.9	10.3

The extraction can be parallelized after extracting the embedding permutation, and it can be done in the memory without requiring GPU resources. Nevertheless, the construction of the cost matrices for the linear assignment problem can be accelerated using GPUs, which requires much less memory than loading the model itself.

In Table 6, we report the average time cost of inserting and extracting watermarks for different model sizes. We include two additional models, Stable LM-2-12B and Llama2-13B, to demonstrate the cost for larger models. As a comparison, we also report the time cost of the baseline method in Table 7. All results are averaged over 10 runs on a platform with an AMD EPYC 7T83 CPU (using 25 cores), 100 GB memory, and an NVIDIA L40 GPU.

Evaluation results show that both methods are efficient in the insertion process and the proposed method is about $3 \sim 12$ times faster in the extraction process. This is because our proposed linear assignment-based permutation extraction algorithm has a better time complexity than brute-force search, especially when the search space is large. The efficient insertion and extraction process makes the proposed method a lightweight watermarking approach for large transformer models.

8 Discussion

Integration of Other FITs and ECCs. It is possible to integrate ECCs with other FITs, such as SU and IM. However, continuous parameters of these transformations (e.g., scaling vectors) are often more sensitive to numerical perturbations than discrete parameters (i.e., permutations). This could complicate the design of the encoding mechanism and the robust extraction algorithm. Therefore, we focus on combining ECCs with weight permutations (WP).

Limitations. The proposed watermarking scheme has several limitations. Firstly, it is restricted to white-box settings, where the provider must have full access to the weights of both the original model and the distributed model. Secondly, although the proposed method is more robust than existing methods, it still cannot prevent the adversary from removing the watermark if the corruption exceeds the correction ability of the ECC. Additionally, the design of the watermarking scheme only provides a theoretical guarantee against adaptive attacks based on FITs. Sophisticated attacks that exploit the specific structure of the watermarking scheme may still be possible. For example, model merging [72] was shown to be effective against existing fingerprinting methods and quantization-based watermarking methods [11]. This technique could also be used to attack the proposed method, as the permutation structures are likely to be disrupted during the merging process.

Generalization to Larger Models. With the rapid development of large language models, the architectures and techniques they employ have become increasingly diverse. However, their essence remains rooted in the transformer architecture. Therefore, the proposed watermarking scheme remains applicable. Moreover, the larger the model, the more attention blocks it has, and the higher the correction ability of the watermark, which makes it more difficult for the adversary to remove the watermark in larger models.

Attacking Generalized Reed-Solomon Codes. The Sidelnikov Shestakov attack [58] is a well-known attack on Reed-Solomon codes. It exploits the Vandermonde structure of Reed-Solomon generator matrices to recover the private key parameters. When evaluation points (*i.e.*, parameters $\boldsymbol{e}$) are consecutive integers, certain rows can be expressed as low-degree polynomial combinations, and the cost of the attack is significantly reduced [46]. To mitigate this, [23] proposed randomizing the parity check matrix to make the attack more difficult. Such a defense mechanism can be incorporated into the proposed watermarking scheme to enhance its security. Additionally, using a larger finite field size (*e.g.*, $q = 2^{24}$) can increase the difficulty of the attack.

Dynamic Watermark Renewal. The lightweight nature of the proposed watermarking scheme enables dynamic updates of embedded identifiers, which enhances privacy protection in real-world deployment. Specifically, model providers can issue temporary tokens (*e.g.*, time-bound cryptographic keys) as watermarks and refresh them periodically. This mechanism mitigates risks such as long-term identifier tracking (where adversaries correlate multiple model copies to infer ownership patterns) and adaptive reverse-engineering attacks. Providers can either reconfigure the generalized Reed-Solomon parameters or adopt distinct rank/unrank functions for permutation mapping across update cycles. Such flexibility forces adversaries to restart codebook analysis after each update, significantly increasing the difficulty of adaptive attacks.

9 Related Work

9.1 Neural Network Watermarking

Model watermarking serves as a passive intellectual property protection technique that embeds a unique identifier into a model without significantly affecting its performance. The watermark can be used to prove ownership, detect unauthorized use, or trace the provenance of the model [19]. Traditional white-box watermarking methods [13, 65, 69] exploit the over-parameterization of neural

networks to embed secret signatures into model weights. These methods often rely on regularization or optimization techniques that require training. Additionally, [45] introduced Invariant Neuron Transformations (INTs) to remove white-box watermarks in convolutional neural networks (CNNs). These transformations shuffle and scale model weights while preserving functionality, which are essentially the same as the weight permutation (WP) and scaling/unscaling (SU) transformations explored in [18] and this work. However, they are designed for watermark removal rather than embedding. Consequently, our proposed method can also resist INT-based attacks. Black-box watermarking methods [1, 25], on the other hand, plant a backdoor as a watermark during the training phase. These watermarks can be extracted by querying the model with trigger inputs. While avoiding the need for white-box access during extraction, these methods typically require significant computational overhead for embedding. The robustness of black-box watermarking methods has also been questioned, as they can sometimes be removed by subsequent fine-tuning [2, 54].

Prominent watermarking methods for LLMs leverage backdoor-based techniques [47, 56, 71]. These methods typically require fine-tuning to embed backdoors into the model that can be triggered by specific inputs, which incurs much higher computational overhead than our approach. Moreover, our approach is developed with theoretical considerations for robustness against certain adaptive attacks, offering guarantees that are often only empirically evaluated in existing methods. Another line of research aims to watermark quantized LLMs during the quantization process [34, 77], which is computationally efficient but not applicable to all models. Our approach, which is different from these methods, is built on the foundation of the FIT-based watermarking method [18] that is training-free and applicable to all transformer models. However, adversaries can exploit the same transformation to remove or forge FIT-based watermarks, making them vulnerable to adaptive attacks. The integration of ECCs into our watermarking scheme enhances its robustness against such attacks.

Watermarking LLM-generated content is another active research area, primarily focusing on distinguishing AI-generated output (e.g., text [10, 14, 32], images [66]) from human-created content. These approaches usually embed watermarks by modifying the model's generation process or altering character encodings. Notably, [10] proposed a novel pseudorandom ECC to watermark generated text, which is similar to our approach but focuses on the generation process rather than the model itself.

9.2 Neural Network Model Copyright Protection

Model fingerprinting [6, 40], another kind of passive intellectual property protection technique, does not require additional tuning or modification of the model that could tamper with its performance. Instead, it uses the inherent and unique characteristics [48] of the model to generate a fingerprint that can be used to identify the model. The fingerprint can still be extracted even after extensive modifications to the model [9, 35], such as fine-tuning or pruning. Therefore, model fingerprinting can be used to trace the origin of a model, even if the model has been altered. However, fingerprints cannot be used to identify a specific distributed model since they are inherent to the model and do not change with distribution [19]. Watermarking methods, on the other hand, can embed a unique identifier for each distributed model, allowing the model provider to trace the origin of each model individually. These two techniques complement each other and can be used together to provide comprehensive protection for the model.

Active protection techniques differ from passive protection techniques in that they aim to prevent copyright infringement from occurring. Model authentication [7, 8, 37] manages the authorized usage of the model by encoding it with a secret key, so that only authorized users with valid keys can access the model normally, while unauthorized users can only use the model with reduced performance. Another active protection technique is inference perturbation [30, 31, 33], which defends against model extraction attacks by modifying the inference results to prevent adversaries from stealing the model.

9.3 Privacy Threats in Deep Learning

Existing threats to privacy in deep learning can be classified into three main categories: membership inference, model inversion and attribute inference, and model extraction. Membership inference attacks [36, 38, 57] speculate whether a specific data sample was used to train a given target model, thereby exposing sensitive information patterns. Model inversion and attribute inference attacks [22, 70, 73] aim to reconstruct sensitive input features or infer hidden attributes, directly compromising individual privacy. Model extraction attacks [28, 64, 75] enable adversaries to steal model parameters or functionalities through black-box queries, potentially resulting in a privacy breach. Privacy-preserving techniques to address these privacy threats mainly include data anonymization, differential privacy [68], homomorphic encryption [24, 53], and secure multi-party computation [51, 52].

10 Conclusion

In this paper, we integrate error correction codes with weight permutations to develop a novel watermarking scheme for large language models. The proposed approach is training-free and has a lower computational overhead compared to existing FIT-based watermarking methods. We utilize generalized Reed-Solomon codes to encode the raw model identifier, transforming it into a sequence of permutations with no fixed points. This strategy effectively safeguards against watermark removal and forgery attacks. We further demonstrate the method's resistance against various FIT attacks and provide a probability analysis for detecting corruption in the watermark. The extensive evaluation results showcase the robustness of our proposed watermarking method against various attacks with low computational overhead. Overall, our approach provides a scalable and secure solution for safeguarding the intellectual property of large language models.

Acknowledgments

This work was supported by the National Natural Science Foundation under Grant No. 62172019 and the Beijing Natural Science Foundation under Grant Nos. QY23041, QY24035.

References

[1] Yossi Adi, Carsten Baum, Moustapha Cisse, Benny Pinkas, and Joseph Keshet. 2018. Turning Your Weakness Into a Strength: Watermarking Deep Neural Networks by Backdooring. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, Baltimore, MD, USA, 1615–1631.

[2] William Aiken, Hyoungshick Kim, Simon Woo, and Jungwoo Ryoo. 2021. Neural Network Laundering: Removing Black-Box Backdoor Watermarks from Deep Neural Networks. *Computers & Security* 106 (July 2021), 102277. https://doi.org/10.1016/j.cose.2021.102277

[3] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebron, and Sumit Sanghai. 2023. GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints. In *The 2023 Conference on Empirical Methods in Natural Language Processing*. https://openreview.net/forum?id=hmOwOZWzYE

[4] Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. 2021. Program Synthesis with Large Language Models. arXiv:2108.07732 [cs.PL] https://arxiv.org/abs/2108.07732

[5] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E. Hinton. 2016. Layer Normalization. https://doi.org/10.48550/arXiv.1607.06450 arXiv:1607.06450 [stat.ML]

[6] Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. 2021. IPGuard: Protecting Intellectual Property of Deep Neural Networks via Fingerprinting the Classification Boundary. In *Proceedings of the 2021 ACM Asia Conference on Computer and Communications Security (ASIA CCS '21)*. Association for Computing Machinery, New York, NY, USA, 14–25. https://doi.org/10.1145/3433210.3437526

[7] Abhishek Chakraborty, Ankit Mondai, and Ankur Srivastava. 2020. Hardware-Assisted Intellectual Property Protection of Deep Learning Models. In *2020 57th ACM/IEEE Design Automation Conference (DAC)*. 1–6. https://doi.org/10.1109/DAC18072.2020.9218651

[8] Huili Chen, Cheng Fu, Bita Darvish Rouhani, Jishen Zhao, and Farinaz Koushanfar. 2019. DeepAttest: An End-to-End Attestation Framework for Deep Neural Networks. In *2019 ACM/IEEE 46th Annual International Symposium on Computer Architecture (ISCA)*, Srilatha Bobbie Manne, Hillery C. Hunter, and Erik R. Altman (Eds.). 487–498. https://doi.org/10.1145/3307650.3322251

[9] Jialuo Chen, Jingyi Wang, Tinglan Peng, Youcheng Sun, Peng Cheng, Shouling Ji, Xingjun Ma, Bo Li, and Dawn Song. 2022. Copy, Right? A Testing Framework for Copyright Protection of Deep Learning Models. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 824–841. https://doi.org/10.1109/SP46214.2022.9833747

[10] Miranda Christ and Sam Gunn. 2024. Pseudorandom Error-Correcting Codes. In *Advances in Cryptology – CRYPTO 2024: 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18–22, 2024, Proceedings, Part VI* (Santa Barbara, CA, USA). Springer-Verlag, Berlin, Heidelberg, 325–347. https://doi.org/10.1007/978-3-031-68391-6_10

[11] Tianshuo Cong, Delong Ran, Zesen Liu, Xinlei He, Jinyuan Liu, Yichen Gong, Qi Li, Anyu Wang, and Xiaoyun Wang. 2024. Have You Merged My Model? On The Robustness of Large Language Model IP Protection Methods Against Model Merging. In *Proceedings of the 1st ACM Workshop on Large AI Systems and Models with Privacy and Safety Analysis* (Salt Lake City, UT, USA) *(LAMPS '24)*. Association for Computing Machinery, New York, NY, USA, 69–76. https://doi.org/10.1145/3689217.3690614

[12] Cyberspace Administration of China. 2023. Interim Measures for the Management of Generative Artificial Intelligence Services. https://www.cac.gov.cn/2023-07/13/c_1690898327029107.htm Accessed: 2025-02-25.

[13] Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. 2019. DeepSigns: An End-to-End Watermarking Framework for Ownership Protection of Deep Neural Networks. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '19)*. Association for Computing Machinery, New York, NY, USA, 485–497. https://doi.org/10.1145/3297858.3304051

[14] Sumanth Dathathri, Abigail See, Sumedh Ghaisas, Po-Sen Huang, Rob McAdam, Johannes Welbl, Vandana Bachani, Alex Kaskasoli, Robert Stanforth, Tatiana Matejovicova, Jamie Hayes, Nidhi Vyas, Majd Al Merey, Jonah Brown-Cohen, Rudy Bunel, Borja Balle, Taylan Cemgil, Zahra Ahmed, Kitty Stacpoole, Ilia Shumailov, Ciprian Baetu, Sven Gowal, Demis Hassabis, and Pushmeet Kohli. 2024. Scalable watermarking for identifying large language model outputs. *Nature* 634 (Oct. 2024), 818–823. https://doi.org/10.1038/s41586-024-08025-4

[15] DeepSeek-AI. 2025. DeepSeek-V3 Technical Report. arXiv:2412.19437 [cs.CL] https://arxiv.org/abs/2412.19437

[16] Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. 2022. GLM: General Language Model Pretraining with Autoregressive Blank Infilling. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (Eds.). Association for Computational Linguistics, Dublin, Ireland, 320–335. https://doi.org/10.18653/v1/2022.acl-long.26

[17] European Union. 2024. Artificial Intelligence Act (EU AI Act). https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32024R1689 Accessed: 2025-02-25.

[18] Pierre Fernandez, Guillaume Couairon, Teddy Furon, and Matthijs Douze. 2024. Functional Invariants To Watermark Large Transformers. In *ICASSP 2024 - 2024 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 4815–4819. https://doi.org/10.1109/ICASSP48485.2024.10447264

[19] Alaa Fkirin, Gamal Attiya, Ayman El-Sayed, and Marwa A Shouman. 2022. Copyright protection of deep neural network models using digital watermarking: a comparative study. *Multimedia Tools and Applications* 81, 11 (2022), 15961–15975.

[20] Elias Frantar and Dan Alistarh. 2023. SparseGPT: massive language models can be accurately pruned in one-shot. In *Proceedings of the 40th International Conference on Machine Learning* (Honolulu, Hawaii, USA) *(ICML'23)*. JMLR.org, Article 414, 15 pages.

[21] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. 2023. OPTQ: Accurate Quantization for Generative Pre-trained Transformers. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=tcbBPnfwxS

[22] Matthew Fredrikson, Eric Lantz, Somesh Jha, Simon Lin, David Page, and Thomas Ristenpart. 2014. Privacy in pharmacogenetics: an end-to-end case study of personalized warfarin dosing. In *Proceedings of the 23rd USENIX Conference on Security Symposium* (San Diego, CA) *(SEC'14)*. USENIX Association, USA, 17–32.

[23] Ernst M. Gabidulin and Olaf Kjelsen. 1994. How to avoid the Sidel'nikov-Shestakov attack. In *Error Control, Cryptology, and Speech Compression*, Andrew Chmora and Stephen B. Wicker (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 25–32.

[24] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *Proceedings of The 33rd International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 48)*, Maria Florina Balcan and Kilian Q. Weinberger (Eds.). PMLR, New York, New York, USA, 201–210.

[25] Tianyu Gu, Kang Liu, Brendan Dolan-Gavitt, and Siddharth Garg. 2019. BadNets: Evaluating Backdooring Attacks on Deep Neural Networks. *IEEE Access* 7 (2019), 47230–47244. https://doi.org/10.1109/ACCESS.2019.2909068

[26] Dan Hendrycks and Kevin Gimpel. 2023. Gaussian Error Linear Units (GELUs). https://doi.org/10.48550/arXiv.1606.08415 arXiv:1606.08415 [cs.LG]

[27] Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*. https://openreview.net/forum?id=nZeVKeeFYf9

[28] Hailong Hu and Jun Pang. 2021. Stealing Machine Learning Models: Attacks and Countermeasures for Generative Adversarial Networks. In *Proceedings of the 37th Annual Computer Security Applications Conference* (Virtual Event, USA) *(ACSAC '21)*. Association for Computing Machinery, New York, NY, USA, 1–16. https://doi.org/10.1145/3485832.3485838

[29] Roy Jonker and Ton Volgenant. 1987. A shortest augmenting path algorithm for dense and sparse linear assignment problems. *Computing* 38 (1987), 325–340. https://doi.org/10.1007/BF02278710

[30] Mika Juuti, Sebastian Szyller, Samuel Marchal, and N. Asokan. 2019. PRADA: Protecting Against DNN Model Stealing Attacks. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE Computer Society, Los Alamitos, CA, USA, 512–527. https://doi.org/10.1109/EuroSP.2019.00044

[31] Sanjay Kariyappa and Moinuddin K. Qureshi. 2020. Defending Against Model Stealing Attacks With Adaptive Misinformation. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Computer Vision Foundation / IEEE, 767–775. https://doi.org/10.1109/CVPR42600.2020.00085

[32] John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. 2024. A Watermark for Large Language Models. arXiv:2301.10226 [cs.LG] https://arxiv.org/abs/2301.10226

[33] Taesung Lee, Benjamin Edwards, Ian M. Molloy, and Dong Su. 2019. Defending Against Neural Network Model Stealing Attacks Using Deceptive Perturbations. In *2019 IEEE Security and Privacy Workshops (SPW)*. IEEE, 43–49. https://doi.org/10.1109/SPW.2019.00020

[34] Linyang Li, Botian Jiang, Pengyu Wang, Ke Ren, Hang Yan, and Xipeng Qiu. 2023. Watermarking LLMs with Weight Quantization. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 3368–3378. https://doi.org/10.18653/v1/2023.findings-emnlp.220

[35] Yuanchun Li, Ziqi Zhang, Bingyan Liu, Ziyue Yang, and Yunxin Liu. 2021. ModelDiff: testing-based DNN similarity comparison for model reuse detection. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2021)*. Association for Computing Machinery, New York, NY, USA, 139–151. https://doi.org/10.1145/3460319.3464816

[36] Zheng Li and Yang Zhang. 2021. Membership Leakage in Label-Only Exposures. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security* (Virtual Event, Republic of Korea) *(CCS '21)*. Association for Computing Machinery, New York, NY, USA, 880–895. https:

//doi.org/10.1145/3460120.3484575
[37] Ning Lin, Xiaoming Chen, Hang Lu, and Xiaowei Li. 2021. Chaotic Weights: A Novel Approach to Protect Intellectual Property of Deep Neural Networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 40, 7 (2021), 1327–1339. https://doi.org/10.1109/TCAD.2020.3018403
[38] Yunhui Long, Vincent Bindschaedler, and Carl A. Gunter. 2017. Towards Measuring Membership Privacy. arXiv:1712.09136 [cs.CR] https://arxiv.org/abs/1712.09136
[39] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. In *International Conference on Learning Representations*. https://openreview.net/forum?id=Bkg6RiCqY7
[40] Nils Lukas, Yuxuan Zhang, and Florian Kerschbaum. 2021. Deep Neural Network Fingerprinting by Conferrable Adversarial Examples. In *International Conference on Learning Representations*. https://openreview.net/forum?id=VqzVhqxkjH1
[41] Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer Sentinel Mixture Models. arXiv:1609.07843 [cs.CL]
[42] Kenji Mikawa and Ken Tanaka. 2023. Efficient linear-time ranking and unranking of derangements. *Inform. Process. Lett.* 179, C (Jan. 2023), 8 pages. https://doi.org/10.1016/j.ipl.2022.106288
[43] Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2023. CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=iaYcJKpY2B_
[44] OpenAI. 2024. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL] https://arxiv.org/abs/2303.08774
[45] Xudong Pan, Mi Zhang, Yifan Yan, Yining Wang, and Min Yang. 2023. Cracking White-box DNN Watermarks via Invariant Neuron Transforms. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining* (Long Beach, CA, USA) *(KDD '23)*. Association for Computing Machinery, New York, NY, USA, 1783–1794. https://doi.org/10.1145/3580305.3599291
[46] Dingyi Pei and Jingang Liu. 2019. A Note on the Sidelnikov-Shestakov Attack of Niederreiter Scheme. In *Information Security and Cryptology*, Fuchun Guo, Xinyi Huang, and Moti Yung (Eds.). Springer International Publishing, Cham, 621–625.
[47] Wenjun Peng, Jingwei Yi, Fangzhao Wu, Shangxi Wu, Bin Bin Zhu, Lingjuan Lyu, Binxing Jiao, Tong Xu, Guangzhong Sun, and Xing Xie. 2023. Are You Copying My Model? Protecting the Copyright of Large Language Models for EaaS via Backdoor Watermark. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 7653–7668. https://doi.org/10.18653/v1/2023.acl-long.423
[48] Zirui Peng, Shaofeng Li, Guoxing Chen, Cheng Zhang, Haojin Zhu, and Minhui Xue. 2022. Fingerprinting Deep Neural Networks Globally via Universal Adversarial Perturbations. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 13420–13429. https://doi.org/10.1109/CVPR52688.2022.01307
[49] Ofir Press and Lior Wolf. 2017. Using the Output Embedding to Improve Language Models. https://doi.org/10.48550/arXiv.1608.05859 arXiv:1608.05859 [cs.CL]
[50] pzc163. 2024. Project author team stay tuned: I found out that the llama3-V project is stealing a lot of academic work from MiniCPM-Llama3-V 2.5. GitHub Issue. https://github.com/OpenBMB/MiniCPM-o/issues/196 Accessed: 2025-02-25.
[51] M. Sadegh Riazi, Mohammad Samragh, Hao Chen, Kim Laine, Kristin Lauter, and Farinaz Koushanfar. 2019. XONN: XNOR-based oblivious deep neural network inference. In *Proceedings of the 28th USENIX Conference on Security Symposium* (Santa Clara, CA, USA) *(SEC'19)*. USENIX Association, USA, 1501–1518.
[52] Bita Darvish Rouhani, M. Sadegh Riazi, and Farinaz Koushanfar. 2018. Deepsecure: scalable provably-secure deep learning. In *Proceedings of the 55th Annual Design Automation Conference* (San Francisco, California) *(DAC '18)*. Association for Computing Machinery, New York, NY, USA, Article 2, 6 pages. https://doi.org/10.1145/3195970.3196023
[53] Amartya Sanyal, Matt Kusner, Adria Gascon, and Varun Kanade. 2018. TAPAS: Tricks to Accelerate (encrypted) Prediction As a Service. In *Proceedings of the 35th International Conference on Machine Learning (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer Dy and Andreas Krause (Eds.). PMLR, 4490–4499.
[54] Masoumeh Shafieinejad, Nils Lukas, Jiaqi Wang, Xinda Li, and Florian Kerschbaum. 2021. On the Robustness of Backdoor-based Watermarking in Deep Neural Networks. In *Proceedings of the 2021 ACM Workshop on Information Hiding and Multimedia Security (IH&MMSec '21)*. Association for Computing Machinery, New York, NY, USA, 177–188. https://doi.org/10.1145/3437880.3460401
[55] Noam Shazeer. 2020. GLU Variants Improve Transformer. *CoRR* abs/2002.05202 (2020). arXiv:2002.05202
[56] Anudeex Shetty, Yue Teng, Ke He, and Qiongkai Xu. 2024. WARDEN: Multi-Directional Backdoor Watermarks for Embedding-as-a-Service Copyright Protection. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 13430–13444. https://doi.org/10.18653/v1/2024.acl-long.725
[57] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership Inference Attacks Against Machine Learning Models. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE Computer Society, Los Alamitos, CA, USA, 3–18. https://doi.org/10.1109/SP.2017.41
[58] V. M. Sidelnikov and S. O. Shestakov. 1992. On insecurity of cryptosystems based on generalized Reed-Solomon codes. *Discrete Mathematics and Applications* 2, 4 (1992), 439–444. https://doi.org/doi:10.1515/dma.1992.2.4.439
[59] Jianlin Su, Yu Lu, Shengfeng Pan, Bo Wen, and Yunfeng Liu. 2021. RoFormer: Enhanced Transformer with Rotary Position Embedding. *CoRR* abs/2104.09864 (2021). arXiv:2104.09864
[60] Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. 2024. A Simple and Effective Pruning Approach for Large Language Models. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=PxoFut3dWW
[61] Yu Sun, Shuohuan Wang, Shikun Feng, Siyu Ding, Chao Pang, Junyuan Shang, Jiaxiang Liu, Xuyi Chen, Yanbin Zhao, Yuxiang Lu, Weixin Liu, Zhihua Wu, Weibao Gong, Jianzhong Liang, Zhizhou Shang, Peng Sun, Wei Liu, Xuan Ouyang, Dianhai Yu, Hao Tian, Hua Wu, and Haifeng Wang. 2021. ERNIE 3.0: Large-scale Knowledge Enhanced Pre-training for Language Understanding and Generation. arXiv:2107.02137 [cs.CL] https://arxiv.org/abs/2107.02137
[62] The White House. 2023. Executive Order 14110: Safe, Secure, and Trustworthy Development and Use of Artificial Intelligence. https://www.federalregister.gov/documents/2023/11/01/2023-24283/safe-secure-and-trustworthy-development-and-use-of-artificial-intelligence Accessed: 2025-02-25.
[63] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. LLaMA: Open and Efficient Foundation Language Models. arXiv:2302.13971 [cs.CL] https://arxiv.org/abs/2302.13971
[64] Florian Tramèr, Fan Zhang, Ari Juels, Michael K. Reiter, and Thomas Ristenpart. 2016. Stealing machine learning models via prediction APIs. In *Proceedings of the 25th USENIX Conference on Security Symposium* (Austin, TX, USA) *(SEC'16)*. USENIX Association, USA, 601–618.
[65] Yusuke Uchida, Yuki Nagai, Shigeyuki Sakazawa, and Shin'ichi Satoh. 2017. Embedding Watermarks into Deep Neural Networks. In *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval (ICMR '17)*. Association for Computing Machinery, New York, NY, USA, 269–277. https://doi.org/10.1145/3078971.3078974
[66] Shan Wan, Wu Liu, Yijun Liu, Feiniu Yuan, and Chunli Meng. 2024. Watermarking Vision-Language Models. In *Proceedings of the 6th ACM International Conference on Multimedia in Asia (MMAsia '24)*. Association for Computing Machinery, New York, NY, USA, Article 119, 1 pages. https://doi.org/10.1145/3696409.3700483
[67] Haiming Wang, Huajian Xin, Chuanyang Zheng, Zhengying Liu, Qingxing Cao, Yinya Huang, Jing Xiong, Han Shi, Enze Xie, Jian Yin, Zhenguo Li, and Xiaodan Liang. 2024. LEGO-Prover: Neural Theorem Proving with Growing Libraries. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=3f5PALef5B
[68] Ji Wang, Jianguo Zhang, Weidong Bao, Xiaomin Zhu, Bokai Cao, and Philip S. Yu. 2018. Not Just Privacy: Improving Performance of Private Deep Learning in Mobile Cloud. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining* (London, United Kingdom) *(KDD '18)*. Association for Computing Machinery, New York, NY, USA, 2407–2416. https://doi.org/10.1145/3219819.3220106
[69] Tianhao Wang and Florian Kerschbaum. 2021. RIGA: Covert and Robust White-Box Watermarking of Deep Neural Networks. In *Proceedings of the Web Conference 2021* (Ljubljana, Slovenia) *(WWW '21)*. Association for Computing Machinery, New York, NY, USA, 993–1004. https://doi.org/10.1145/3442381.3450000
[70] Xi Wu, Matthew Fredrikson, Somesh Jha, and Jeffrey F. Naughton. 2016. A Methodology for Formalizing Model-Inversion Attacks. In *2016 IEEE 29th Computer Security Foundations Symposium (CSF)*. 355–370. https://doi.org/10.1109/CSF.2016.32
[71] Jiashu Xu, Fei Wang, Mingyu Ma, Pang Wei Koh, Chaowei Xiao, and Muhao Chen. 2024. Instructional Fingerprinting of Large Language Models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, Kevin Duh, Helena Gomez, and Steven Bethard (Eds.). Association for Computational Linguistics, Mexico City, Mexico, 3277–3306. https://doi.org/10.18653/v1/2024.naacl-long.180
[72] Enneng Yang, Li Shen, Guibing Guo, Xingwei Wang, Xiaochun Cao, Jie Zhang, and Dacheng Tao. 2024. Model Merging in LLMs, MLLMs, and Beyond: Methods, Theories, Applications and Opportunities. arXiv:2408.07666 [cs.LG] https://arxiv.org/abs/2408.07666
[73] Samuel Yeom, Irene Giacomelli, Matt Fredrikson, and Somesh Jha. 2018. Privacy Risk in Machine Learning: Analyzing the Connection to Overfitting. In *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*. IEEE Computer Society, Los Alamitos, CA, USA, 268–282. https://doi.org/10.1109/CSF.2018.00027
[74] Longhui Yu, Weisen Jiang, Han Shi, Jincheng YU, Zhengying Liu, Yu Zhang, James Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2024. MetaMath:

Bootstrap Your Own Mathematical Questions for Large Language Models. In *The Twelfth International Conference on Learning Representations*. https://openreview.net/forum?id=N8N0hgNDRt

[75] Xiaoyong Yuan, Leah Ding, Lan Zhang, Xiaolin Li, and Dapeng Oliver Wu. 2022. ES Attack: Model Stealing Against Deep Neural Networks Without Data Hurdles. *IEEE Transactions on Emerging Topics in Computational Intelligence* 6, 5 (2022), 1258–1270. https://doi.org/10.1109/TETCI.2022.3147508

[76] Biao Zhang and Rico Sennrich. 2019. Root Mean Square Layer Normalization. https://doi.org/10.48550/arXiv.1910.07467 arXiv:1910.07467 [cs.LG]

[77] Ruisi Zhang and Farinaz Koushanfar. 2024. EmMark: Robust Watermarks for IP Protection of Embedded Quantized Large Language Models. In *Proceedings of the 61st ACM/IEEE Design Automation Conference* (San Francisco, CA, USA) *(DAC '24)*. Association for Computing Machinery, New York, NY, USA, Article 88, 6 pages. https://doi.org/10.1145/3649329.3655674

A Permutation Extraction

In this section, we provide the complete proof of the permutation extraction algorithm and the theoretical analysis of the permutation extraction process.

Theorem A.1. *Under the assumption of Eq.* (17)*, the permutation π that satisfies Eq.* (16) *is the unique optimal solution to the following linear assignment problem:*

$$\pi = \arg\min_{\sigma\in\Pi_d} \sum_{i=1}^{d} cost(i,\sigma(i)) = \arg\min_{\sigma\in\Pi_d} \sum_{i=1}^{d} \|W_{:,i}{}' - W_{:,\sigma(i)}\|_2, \quad (30)$$

where $cost(i,\sigma(i))$ is the cost of matching the i-th column of W with the $\sigma(i)$-th column of W'.

Proof. The cost of π is given by

$$\sum_{i=1}^{d} \|W_{:,i}{}' - W_{:,\pi(i)}\|_2 = \sum_{i=1}^{d} \|(WC_\pi)_{:,i} - W_{:,\pi(i)} + \epsilon_{:,i}\|_2 = \sum_{i=1}^{d} \|\epsilon_{:,i}\|_2. \quad (31)$$

For any $\sigma \in \Pi_d$ that is not π, let $\mathcal{I} = \{i \mid \pi(i) \neq \sigma(i)\}$, then we have:

$$\sum_{i=1}^{d} \|W_{:,i}{}' - W_{:,\sigma(i)}\|_2 - \sum_{i=1}^{d} \|W_{:,i}{}' - W_{:,\pi(i)}\|_2 \quad (32)$$

$$= \sum_{k\in\mathcal{I}} \left(\|W_{:,\pi(k)} - W_{:,\sigma(k)} + \epsilon_{:,k}\|_2 - \|\epsilon_{:,k}\|_2\right) \quad (33)$$

$$\geq \sum_{k\in\mathcal{I}} \left(\|W_{:,\pi(k)} - W_{:,\sigma(k)}\|_2 - 2\|\epsilon_{:,k}\|_2\right). \quad (34)$$

According to Eq. (17), we have $\|W_{:,\pi(k)} - W_{:,\sigma(k)}\|_2 \geq 2\delta > 2\|\epsilon_{:,k}\|_2$ for all $k \in \mathcal{I}$, where the last inequality is strict. Therefore, π is the unique optimal solution to the linear assignment problem. □

Theorem A.2. *Under the assumption of Eq.* (17)*, if the permutation that satisfies Eq.* (16) *is a block permutation $\pi^b \in \Pi_d^b$, then π is the unique optimal solution to the following linear assignment problem:*

$$\pi = \arg\min_{\sigma\in\Pi_{d/b}} \sum_{i=1}^{d/b} cost_{blk}(i,\sigma(i)). \quad (35)$$

The block permutation can be obtained by $\pi^b = \mathrm{BP}(\pi; b)$.

Proof. Similarly, the cost of π is given by

$$\sum_{i=1}^{d/b}\sum_{j=1}^{b} \|W_{:,i\cdot b+j}{}' - W_{:,\pi(i)\cdot b+j}\|_2 \quad (36)$$

$$= \sum_{i=1}^{d/b}\sum_{j=1}^{b} \|W_{:,i\cdot b+j}C_{\pi^b} - W_{:,\pi(i)\cdot b+j} + \epsilon_{:,i\cdot b+j}\|_2 \quad (37)$$

$$= \sum_{i=1}^{d} \|\epsilon_{:,i}\|_2. \quad (38)$$

For any $\sigma \in \Pi_{d/b}$ that is not π, let $\mathcal{I} = \{i \mid \pi(i) \neq \sigma(i)\}$, then we have:

$$\sum_{i=1}^{d/b}\sum_{j=1}^{b} \|W_{:,i\cdot b+j}{}' - W_{:,\sigma(i)\cdot b+j}\|_2 - \sum_{i=1}^{d/b}\sum_{j=1}^{b} \|W_{:,i\cdot b+j}{}' - W_{:,\pi(i)\cdot b+j}\|_2 \quad (39)$$

$$= \sum_{k\in\mathcal{I}}\sum_{j=1}^{b} \left(\|W_{:,\pi(k)\cdot b+j} - W_{:,\sigma(k)\cdot b+j} + \epsilon_{:,k\cdot b+j}\|_2 - \|\epsilon_{:,k\cdot b+j}\|_2\right) \quad (40)$$

$$\geq \sum_{k\in\mathcal{I}}\sum_{j=1}^{b} \left(\|W_{:,\pi(k)\cdot b+j} - W_{:,\sigma(k)\cdot b+j}\|_2 - 2\|\epsilon_{:,k\cdot b+j}\|_2\right). \quad (41)$$

Since $\|W_{:,\pi(k)\cdot b+j} - W_{:,\sigma(k)\cdot b+j}\|_2 \geq 2\delta > 2\|\epsilon_{:,k\cdot b+j}\|_2$ for all $k \in \mathcal{I}$, we similarly conclude that π is the unique optimal solution to the linear assignment problem. □

Theorem A.3. *Given a block diagonal matrix* $\mathrm{diag}(R_1,\cdots,R_n)$*, denoted as R, where each R_i is a 2×2 rotation matrix and has different rotation angles, if an invertible matrix Q satisfies that $QRQ^{-1} = R$, then Q is a block diagonal matrix,* i.e., $\mathrm{diag}(Q_1,\cdots,Q_n)$*, where each Q_i is a 2×2 rotation matrix multiplied by a scalar.*

Proof. Since $QRQ^{-1} = R$, we have $QR = RQ$, which means that Q commutes with R. Denote $Q_{ij} \in \mathbb{R}^{2\times2}$ as the (i,j)-th block of Q, then we have:

$$Q_{ij}R_j = R_iQ_{ij}. \quad (42)$$

When $i \neq j$, $R_i \neq R_j$ as they have different rotation angles, and thus $Q_{ij} = 0$. While for $i = j$, Q_{ii} commutes with R_i, which means that Q_{ii} is a rotation matrix multiplied by a scalar. Therefore, Q is a block diagonal matrix whose blocks are rotation matrices multiplied by scalars. □

Theorem A.4. *Solution of Eq.* (28) *is also the unique optimal solution to the linear assignment problem in Eq.* (18) *after normalizing the rows of W and W', if the normalized matrices satisfies the assumption in Eq.* (16)*, where the normalization is defined as:*

$$\mathrm{normalize}(W) = \mathrm{diag}\left(\frac{1}{\|W_{1,:}\|_2},\cdots,\frac{1}{\|W_{d,:}\|_2}\right)W. \quad (43)$$

Proof. It suffices to show that the normalized matrices satisfy the assumption in Eq. (16). Without loss of generality, we consider the i-th row of W and W', denoted by $W_{i,:}$ and $W_{i,:}{}'$, respectively.

Then we have:

$$W_{i,:}' = \alpha_i W_{i,:} C_\pi + \epsilon_{i,:} \tag{44}$$

$$\text{normalize}(W_{i,:}') = \frac{W_{i,:}'}{\|W_{i,:}'\|_2} \tag{45}$$

$$= \frac{W_{i,:}}{\|W_{i,:}\|_2} C_\pi + \left(\frac{W_{i,:}'}{\|W_{i,:}'\|_2} - \frac{W_{i,:} C_\pi}{\|W_{i,:}\|_2} \right), \tag{46}$$

where the first term is the i-th row of W after normalization, and the second term is the new noise term. To show that the new noise term is efficiently small, we first observe that $W_{i,:}'$ is the result of an affine transformation on $W_{i,:}C_\pi$, whose norm is the same as $W_{i,:}$. Therefore, if we denote $W_{i,:}C_\pi$ as v, then we have:

$$\epsilon_{\text{new}} = \frac{\alpha_i v + \epsilon_{i,:}}{\|\alpha_i v + \epsilon_{i,:}\|_2} - \frac{v}{\|v\|_2}. \tag{47}$$

Under the assumption that $\alpha_i > 0$ and $\|\epsilon_{i,:}\|_2 \ll \|v\|_2$, *i.e.*, the noise is much smaller than the parameter, we have:

$$\|\alpha_i v + \epsilon_{i,:}\|_2 = \sqrt{\alpha_i^2 \|v\|_2^2 + 2\alpha_i \langle v, \epsilon_{i,:} \rangle + \|\epsilon_{i,:}\|_2^2} \tag{48}$$

$$= \alpha_i \|v\|_2 \sqrt{1 + \frac{2\langle v, \epsilon_{i,:} \rangle}{\alpha_i \|v\|_2^2} + \frac{\|\epsilon_{i,:}\|_2^2}{\alpha_i^2 \|v\|_2^2}} \tag{49}$$

$$\approx \alpha_i \|v\|_2 \left(1 + \frac{\langle v, \epsilon_{i,:} \rangle}{\alpha_i \|v\|_2^2} \right). \tag{50}$$

Therefore, the new noise term can be approximated as:

$$\frac{\alpha_i v + \epsilon_{i,:}}{\|\alpha_i v + \epsilon_{i,:}\|_2} - \frac{v}{\|v\|_2} \approx \frac{\alpha_i v + \epsilon_{i,:}}{\alpha_i \|v\|_2 \left(1 + \frac{\langle v, \epsilon_{i,:} \rangle}{\alpha_i \|v\|_2^2} \right)} - \frac{v}{\|v\|_2} \tag{51}$$

$$\approx \frac{\alpha_i v + \epsilon_{i,:}}{\alpha_i \|v\|_2} \left(1 - \frac{\langle v, \epsilon_{i,:} \rangle}{\alpha_i \|v\|_2^2} \right) - \frac{v}{\|v\|_2} \tag{52}$$

$$\approx \frac{\epsilon_{i,:} - u\langle u, \epsilon_{i,:} \rangle}{\alpha_i \|v\|_2}, \tag{53}$$

where u is the unit vector of v. The norm of the new noise term is then bounded by:

$$\|\epsilon_{\text{new}}\|_2 \approx \left\| \frac{\epsilon_{i,:} - u\langle u, \epsilon_{i,:} \rangle}{\alpha_i \|v\|_2} \right\|_2 \tag{54}$$

$$= \frac{1}{\alpha_i \|v\|_2} \sqrt{\|\epsilon_i\|_2^2 - \langle u, \epsilon_i \rangle^2} \tag{55}$$

$$= \frac{\|\epsilon_{i,:}\|_2 \sin\theta}{\alpha_i \|v\|_2} \tag{56}$$

$$< \frac{\delta \sin\theta}{2\alpha_i \|v\|_2}, \tag{57}$$

where θ is the angle between v and $\epsilon_{i,:}$. Consequently, the new noise term is also efficiently small, and as long as the assumption in Eq. (16) holds for the normalized matrices, the solution of Eq. (28) is the unique optimal solution to the linear assignment problem. □

B Corruption Probability Analysis

Theorem B.1. *Suppose the adversary corrupts a derangement $\pi \in \mathcal{D}_d$ by compositing it with a permutation $\sigma \in \Pi_d$, resulting in the extracted permutation $\pi' = \sigma \circ \pi$. Denote the probability that $\phi^{-1}(\pi')$ is defined as $p_{corrupt}$,* i.e., *the corruption is not detected during decoding, then $p_{corrupt} \leq \frac{q}{d!}$ if $\sigma \sim \mathcal{U}(\Pi_d)$.*

Proof. For a permutation σ uniformly sampled from Π_d, the distribution of $\pi' = \sigma \circ \pi$ is also uniform over $\mathcal{D}_d$. Since only q out of $d!$ permutations in $\mathcal{D}_d$ can be decoded by ϕ^{-1}, the probability that $\phi^{-1}(\pi')$ is defined is at most $\frac{q}{d!}$. □

C Settings of Model Modification Techniques

We report the settings of model modification techniques in Section 7 and provide relevant details in this section.

C.1 Quantization

Two quantization methods are used in our experiments: 8-bit static quantization implemented by PyTorch, and AutoGPTQ with 2-bit, 3-bit, and 4-bit quantization. Both quantization methods require a calibration dataset. For this purpose, we use the first 1,000 samples from the WikiText-2 dataset. We follow the default settings of AutoGPTQ for quantization, with group size set to 128 and rows processed based on decreasing activation.

C.2 Pruning

For unstructured pruning, we use two methods, SparseGPT and Wanda, to prune the watermarked models with a target sparsity of 0.5 and 0.7. A calibration dataset is also required for pruning, and we use the same dataset as in the quantization process, except that we only use 128 samples, as recommended by the default settings. For SparseGPT, we set the block size to 128 and Hessian dampening to 0.01. No further hyperparameter for Wanda is required.

C.3 Fine-tuning

We use LoRA to fine-tune watermarked models. The target layers for LoRA include all linear layers except for the output layer. The rank r is set to 12, α is set to 32, and the dropout rate is set to 0.1. We fine-tune using AdamW [39] with a linear learning rate schedule and a batch size of 4 on the WikiText-2 dataset. The learning rate is set to 2e-5 and weight decay is set to 0.01. The first 1,000 steps are used for warm-up with a warmup ratio of 0.1. We stop fine-tuning after a certain amount of tokens have been processed, which is set to 1 million, 5 million, 10 million, 50 million, and 100 million for all models in our evaluation. The training is performed on an NVIDIA L40 GPU with 48 GB of VRAM.

The performance of fine-tuned watermarked Llama models is reported in Table 8, which shows the perplexity (PPL) on the test set of WikiText-2 and the accuracy on the HellaSwag benchmark. We additionally include the performance of the original watermarked models (without fine-tuning) for comparison, as indicated in the Watermarked column. The results show that the perplexity on WikiText-2 slightly decreases after fine-tuning about 1M tokens, while the accuracy on HellaSwag is largely preserved. As the number of fine-tuning tokens increases, the models are overfitted to the training data, leading to a significant increase in perplexity and a decrease in accuracy on HellaSwag.

D Watermark Removal and Forgery Attacks

We repeated 2,000,000 times the watermark removal attack for Llama3.2-1B and Llama3.2-3B under the same settings as in Section 7. Each time, the adversary randomly selects 32 and 56 layers of the model and performs a random permutation on the weights of

Table 8: Performance of fine-tuned watermarked models.

Model	Watermarked		1M Tokens		5M Tokens		10M Tokens		50M Tokens		100M Tokens	
	PPL	HellaSwag	PPL	HellaSwag	PPL	HellaSwag	PPL	HellaSwag	PPL	HellaSwag	PPL	HellaSwag
Llama3.2-1B	8.49	53.55%	8.37	53.64%	8.28	52.21%	109.40	37.02%	58.70	27.49%	1667.03	25.65%
Llama3.2-3B	7.01	61.28%	6.91	61.19%	6.92	61.02%	9.06	37.28%	3679.74	26.25%	418.50	25.29%
Llama2-7B	22.68	61.68%	22.95	61.58%	25.68	60.45%	24.17	59.55%	124.87	52.09%	92797.25	27.60%
Llama3.1-8B	5.97	65.55%	5.97	65.51%	6.39	64.82%	53.13	27.66%	624.44	26.22%	6294.09	25.57%

each selected layer. The number of introduced corruption is within the error correction capability of each watermark. We observed no successful removal of the watermark, and the original identifiers were successfully recovered in all cases.

Table 9 shows the attacking results of watermark forgery. The number of detected permutation corruptions is reported for each model, where the number m/n indicates that there are n corrupted permutations, and m of them are detected and identified as erasures during decoding. The number of decode failures and successes are also reported for each model. If a decode success occurs and the resulting identifier is different from the original one, the forgery attack is considered successful. Note that for Llama3.2-1B and Llama3.2-3B, the forgery attack applies 33 and 57 random permutations, respectively, both exceeding the error correction capability of the watermark.

Table 9: Simulation results of watermark forgery attacks.

Model	Number of detected permutation corruption	Decode failure	Decode success
Llama3.2-1B	66,000,000/66,000,000 (100%)	2,000,000	0
Llama3.2-3B	114,000,000/114,000,000 (100%)	2,000,000	0