FUNCTIONAL INVARIANTS TO WATERMARK LARGE TRANSFORMERS

Pierre Fernandez[1,2⋆], *Guillaume Couairon*[1], *Teddy Furon*[2†], *Matthijs Douze*[1]

[1]FAIR, Meta   [2]Centre Inria de l'Université de Rennes

ABSTRACT

The rapid growth of transformer-based models increases the concerns about their integrity and ownership insurance. Watermarking addresses this issue by embedding a unique identifier into the model, while preserving its performance. However, most existing approaches require to optimize the weights to imprint the watermark signal, which is not suitable at scale due to the computational cost. This paper explores watermarks with virtually no computational cost, applicable to a non-blind white-box setting (assuming access to both the original and watermarked networks). They generate functionally equivalent copies by leveraging the models' invariance, via operations like dimension permutations or scaling/unscaling. This enables to watermark models without any change in their outputs and remains stealthy. Experiments demonstrate the effectiveness of the approach and its robustness against various model transformations (fine-tuning, quantization, pruning), making it a practical solution to protect the integrity of large models.

Index Terms— DNN watermarking, white-box, transformers

1. INTRODUCTION

Large-scale transformer models are a leap forward in the field of machine learning, with large language models like GPT-4 [1], LLaMA [2] and others [3, 4, 5], or vision ones like ViT-22b [6] or DINOv2 [7]. As these models grow in complexity and size, protecting them is important due to investments in their development. Notably, this is raised by the US "Ensuring Safe, Secure, and Trustworthy AI" announcement, European AI Act and Chinese AI governance rules.

Watermarking deep neural networks [8, 9] presents a step towards ensuring their security, integrity and ownership. Embedding a unique identifier into the model enables tracing it to safeguard it from unauthorized usage and distribution. However, watermarking large transformer models poses new challenges. Current watermarking methods involve optimizing the weights to infuse the watermark, either during pre-training or by fine-tuning the weights with additional losses. While these techniques have shown success for smaller models, they become computationally infeasible for large-scale models and for the burgeoning number of potential users and applications.

To address these challenges, we introduce a new approach to watermarking large transformers, when access to both the original and watermarked model is granted, *i.e.* in a non-blind white-box setting. Our method capitalizes on the inherent invariance of transformers. For a given model, it generates equivalent copies that serve as carriers for arbitrary signatures. By employing operations such as dimension permutation and coupled matrix multiplications, we create model replicas without changing the model's outputs and without training. We conduct experiments on state-of-the-art transformer architectures to evaluate the applicability of our approach and its robustness against model processing (*e.g.* fine-tuning, pruning, quantization, etc.). We also discuss the main drawbacks of this setting.

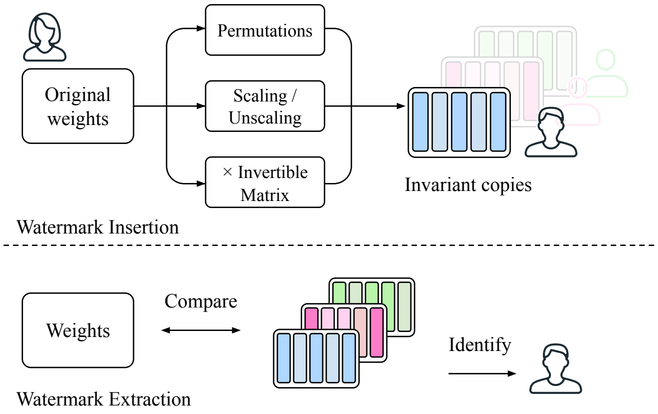

Fig. 1: Overview. We identify each model by applying invariance operations to the original weights.

The paper is organized as follows: section 2 provides an overview of related works on DNN watermarking and background on transformers; section 3 details the transformer's invariants and how to exploit them for watermarking; section 4 presents experimental results on large language models.

Problem statement. A provider *Alice*, distributes her model to various users *Bob* (either individuals or organizations). She aims to trace the model back to a specific user, in case of unauthorized distribution or leaks. As a precautionary measure, Alice embeds a unique signature in the model's weights for each user. In a white-box setting, Alice has access to the models' weights and extracts the signature from it to identify Bob. Besides, Bob may evade detection intentionally (trying to remove the watermark) or unintentionally (fine-tuning, quantization, etc.).

This setting is quite common. Indeed few entities ("Alices") have the necessary computation resources, data and expertise to generate the base model. For example, the training of the 65-B LLaMA model took around 1B GPU-hours. Therefore, there are few variants of such large models in the world. Besides, when Bob gains access to the base model, it is common that he transforms it and that it re-emerges in a public forum or through another channel, so that Alice can analyze it. This can be either because Bob re-distributed it or because Alice sought the model through legal channels, as suggested by Fan *et al.* [10]. For example, many variants of the LLaMA models have been fine-tuned on instruction datasets and been made available online.

⋆Correpondance to pfz@meta.com

†Work supported by ANR / AID under Chaire SAIDA ANR-20-CHIA-0011.

2. RELATED WORK & TECHNICAL BACKGROUND

2.1. Deep Neural Network (DNN) Watermarking

DNN watermarking robustly embeds a unique identifier into the model without affecting its performance, in order to later verify the model's identity. Watermarking should satisfy three criteria, *utility*: the new model should have the same performance as the original one; *security*: it should be stealthy, hard to remove and to forge; *robustness*: the watermark should be detected even after the model has been modified. Modifications may be unintentional – models are fine-tuned, pruned and quantized – or intentional – adversaries may try to remove the watermark or embed their own [11, 12, 13]. For instance, some adversarial transforms employ invariance operations in neurons and ReLU layers to evade detection [14], in a similar fashion as the techniques of this work.

We distinguish between white-box and black-box settings, depending on whether the model weights are accessible at verification time, or only through a remote API. In white-box, the pioneering work [8] embeds watermarks into the DNN's weights. A regularization loss term during training constrains the weights to carry a specific signal, while minimizing the impact on the model's performance. The watermark is then retrieved directly by analyzing the model's weights. The Deep·Signs·Marks [15, 16] extends this to target black-box settings and propose building collusion-resistant watermarks, RIGA [17] improves its covertness and robustness, and greedy residuals [18] improves the selection of the weights to modify.

Another line of work, called trigger-set based methods, embeds the watermark in the behavior of the model with regards to certain inputs. A recurrent idea is to use "backdoors", *i.e.* memorize certain sequences of input-output pairs [9, 19]. Watermarking generative models is also an active field of research, either by employing triggers [20, 21], or by watermarking their outputs [22, 23].

All previous methods require training or optimization. This is expensive computationally and requires extensive expertise in the case of large-scale transformers. The feasibility of existing watermarking methods to these models therefore remains an open question.

2.2. Transformers

Transformer [24] neural networks have become the standard for many applications in the last few years. They can be trained efficiently on GPUs and scale well to large datasets and models, in both natural language processing [25, 26] and computer vision [6, 7]. In the following we describe the NLP architecture from [27].

The input string is first tokenized into a sequence of integers $(x_1, \dots, x_n) \in \mathcal{V}^n$. An embedding layer $E \in \mathbb{R}^{|\mathcal{V}| \times d}$ maps each token x_i to a continuous vector $z_i^0 = E_{x_i} \in \mathbb{R}^d$, where d is the embedding dimension. The transformer is a stack of attention and feed-forward layers, that we describe in the following.

Attention layers. The self-attention mechanism enables long-range dependencies between sequence elements. A self-attention transforms an input sequence $\mathbf{z} \in \mathbb{R}^{n \times d}$ into queries Q, keys K, and values V:

$$Q = \mathbf{z}W^{\mathrm{Q}} \in \mathbb{R}^{n \times d_{\mathrm{k}}};\ K = \mathbf{z}W^{\mathrm{K}} \in \mathbb{R}^{n \times d_{\mathrm{k}}};\ V = \mathbf{z}W^{\mathrm{V}} \in \mathbb{R}^{n \times d_{\mathrm{v}}}. \quad (1)$$

It then computes attention weights by taking a scaled dot product between the queries and keys:

$$\mathrm{Attention}(Q, K, V) = \mathrm{softmax}\left(\frac{QK^\top}{\sqrt{d_{\mathrm{k}}}}\right)V. \quad (2)$$

Where the $\mathrm{Softmax}$ operator is applied column-wise.

This attention operator is applied h times in parallel, yielding h output *heads*. The results are concatenated and projected back to the original dimension:

$$\mathrm{MultiHead}(Q, K, V) = \mathrm{Concat}(\mathrm{head}_1, ., \mathrm{head}_h)W^{\mathrm{O}}, \quad (3)$$

where $\mathrm{head}_i = \mathrm{Attention}(QW_i^{\mathrm{Q}}, KW_i^{\mathrm{K}}, VW_i^{\mathrm{V}})$. The projections $W_i^{\mathrm{Q}}, W_i^{\mathrm{K}} \in \mathbb{R}^{d \times d_{\mathrm{k}}}$, $W_i^{\mathrm{V}} \in \mathbb{R}^{d \times d_{\mathrm{v}}}$ and $W^{\mathrm{O}} \in \mathbb{R}^{hd_{\mathrm{v}} \times d}$ are learned parameters.

The feed-forward network. The output is fed to a feed-forward network (FFN), *e.g.* two linear layers with a ReLU activation:

$$\mathrm{FFN}(\mathbf{h}) = \mathrm{ReLU}(\mathbf{h}W_1 + b_1)W_2 + b_2, \quad (4)$$

where $W_1 \in \mathbb{R}^{d \times d_{\mathrm{ff}}}$, $b_1 \in \mathbb{R}^{d_{\mathrm{ff}}}$, $W_2 \in \mathbb{R}^{d_{\mathrm{ff}} \times d}$ and $b_2 \in \mathbb{R}^d$ are learned parameters (SwiGLU [28] and other variants also frequently replace ReLU).

A stack of residual connections. Instead of directly feeding $\mathbf{z}$ and $\mathbf{h}$ to the attention and FFN layers, residual connections are applied and the inputs are normalized using layer normalization [29] (or variants like RMSnorm [30]):

$$\mathrm{LayerNorm}(\mathbf{z}) = \frac{\mathbf{z} - \mu}{\sigma} \odot g + b, \quad (5)$$

where μ and σ are the mean and standard deviation of $\mathbf{z}$ along its second dimension, and $g \in \mathbb{R}^d$ and $b \in \mathbb{R}^d$ are learned parameters. This is repeated for each layer $l \in \{1, ..., L\}$ of the transformer:

$$\mathbf{h}^l = \mathrm{Att}^l\left(\mathrm{Ln}^l_{\mathrm{att}}(\mathbf{z}^l)\right) + \mathbf{z}^l \quad (6)$$

$$\mathbf{z}^{l+1} = \mathrm{Ffn}^l\left(\mathrm{Ln}^l_{\mathrm{ffn}}(\mathbf{h}^l)\right) + \mathbf{h}^l. \quad (7)$$

The output is fed to a normalization layer $\mathrm{Ln}_{\mathrm{out}}$ and a linear layer $W_{\mathrm{out}} \in \mathbb{R}^{d \times |\mathcal{V}|}$ to generate logits, and a softmax outputs the probability distribution of the next token.

Positional embeddings. For many tasks, it is useful to encode the position of tokens in the input sequence. Positional embeddings are what allows to encode this information. They were originally sinusoidal functions of the position [24] added to the input embeddings. There are now several variants [25, 31, 32, 33], that may change Eq. (2). For instance, rotary embeddings [31] multiply queries and keys depending on their relative position in the sequence. If m is the position of the query ($Q_m = \mathbf{z}_m W^{\mathrm{Q}}$) and n the position of the key, then it rewrites the product of Eq. (2) as:

$$Q_m K_n^\top = z_m W^{\mathrm{Q}} R_{\Theta, n-m} (z_n W^{\mathrm{K}})^\top. \quad (8)$$

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

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

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

3. WATERMARKING THROUGH INVARIANCE

3.1. Invariants in the weights of transformers

We define an invariant as a series of operation applied on the model's weights $\theta \to \theta'$ such that for any input x, the output $f_{\theta'}(x)$ is the same as before the application of the invariant.

Permutation invariance appears in (at least) four levels of the transformer. We note Π^d the set of permutations of $\{1, ..., d\}$. For a matrix $M \in \mathbb{R}^{d\times d}$ and $\pi \in \Pi^d$, we denote by $M_{:,\pi}$ (resp. $M_{\pi,:}$) the matrix where columns (resp. rows) are permuted according to π.

Embedding dimension. The embedding matrix E can be permuted along its second dimension without changing the output of the model, as long as the permutation is propagated to other matrices in the model. More formally, for $\pi \in \Pi^d$, if $E' = E_{:,\pi}$, then matrices $\{W^{\mathrm{Q}}, W^{\mathrm{K}}, W^{\mathrm{V}}, W_1, W_{out}, \mathrm{Ln}_{\mathrm{att}}, \mathrm{Ln}_{\mathrm{ffn}}, b_2\} \subset \theta$ need to be permuted along their first dimension by π and all matrices $\{W^{\mathrm{O}}, W_2\}$ along their second one: $(W^{\mathrm{Q}})' = W^{\mathrm{Q}}_{\pi,:}$, $(W^{\mathrm{O}})' = W^{\mathrm{O}}_{:,\pi}$, etc.

FFN layer dimension. All neurons making up matrices W_1 and W_2 of feed-forward networks can be permuted: for $\pi \in \Pi^{d_{\mathrm{ff}}}$, if $W_1' = (W_1)_{:,\pi}$ and $W_2' = (W_2)_{\pi,:}$, then $f_{\theta'}(\cdot) = f_\theta(\cdot)$.

Attention heads. Because of Eq. 3, heads are interchangeable, provided that W^{O} is permuted in blocks of d_{v} rows, according to its first dimension. In practice, the $(W_i^{\mathrm{Q}})_{i\in\{1,..,h\}}$ are stored as a unique matrix, so permuting the heads is equivalent to permuting the columns of this matrix in blocks of d_{k} (same for W_i^{K} and W_i^{V}).

Inside the head. Depending on the type of positional embeddings, the previous permutations can be extended. For instance if they do not impact Eq. 2 (this is not the case for rotary embeddings) then W^{Q} and W^{K} can be permuted along their second dimension.

Scaling/Unscaling. Whenever layer norms or variants are directly followed (or preceded) by linear layers, *e.g.* at every attention or FFN block, we can rescale component-wise the parameters g, b of Eq. 5 by a vector $\alpha \in \mathbb{R}^d$. Invariance is obtained by dividing the rows of the following (or preceding) linear layers by the same vector.

Invertible matrices in QK products. We hereby assume the positional embeddings do not impact Eq. 2. If $P \in \mathbb{R}^{d_{\mathrm{k}}\times d_{\mathrm{k}}}$ is invertible, then choosing $(W^{\mathrm{Q}})' = W^{\mathrm{Q}}P$ and $(W^{\mathrm{K}})' = W^{\mathrm{K}}(P^\top)^{-1}$ is invariant in Eq. 2. This also applies to the case of rotary embeddings by restricting P to be block diagonal of 2×2 matrices that apply a 2D rotations and scaling by a factor λ (thanks to the commutativity of 2D rotations).

Combining invariants. All previous parameter transformations may be seen as invertible right or left matrix multiplications applied to the model parameters. For instance, π is associated to a permutation matrix P such that $P_{i,\pi(i)} = 1$ and $P_{i,j} = 0$ otherwise, scalings are diagonal matrix, etc. They do not interfere and may be combined in a sequence of arbitrary order, yielding $\theta \to \theta' \to \theta'' \to \cdots$.

Combining transformations at all levels improves robustness to intentional removal attacks and to collusion (*i.e.* when several Bobs share their weights to evade detection). Indeed, if Bob tries to remove the watermark by re-applying one invariant, it will still be present in the other invariants. In the same way, if several Bobs compare their models, it will be hard for them to identify which operations were applied to their models, since the order in which they were applied is unknown, and since the weights will differ a lot between them.

3.2. From invariants to watermarks

Insertion. Before starting the watermark process, for each invariant and each level of the network, we restrict the set of transformations to 2^k. For example, we randomly sample 2^k possible permutations in Π^d for the Embedding dimension (out of the total $d!$). This is to simplify the detection procedure.

Therefore, we can encode k bits for each combination of an invariant and a level. Thus, we encode a model's identifier as the concatenation of m chunks of k bits (2^{mk} possibilities). For instance, let $k = 4$ and the model have 32 layers. We choose to embed two permutations per layer, one for the attention block and one for the FFN block. The total number of bits is $2\times32\times4 = 256$, representing 10^{77} possible models (approximately the number of atoms in the universe, an upper bound of the number of Bobs). This embedding process provides robustness and efficiency when extracting the watermarks.

Extraction. The brute-force way to extract the k-bits message from a weight matrix is to re-apply all 2^k possible invariants to the original matrix. We then compute the Frobenius norm of the difference between the observed weight and the possible watermarked ones. We choose the one with lowest among the 2^k and this choice is encoded as a k-bit integer. Doing that on every blocks of the network and every invariant, we end up with a full message made of m chunks of k bits. In the case of intertwined invariants, we do the same in the order in which we inserted the invariants, reverting them as we go.

To speed up extraction, we may select a subset of the matrices' rows before extraction. This speeds up the extraction (in the order of $100\times$), but makes the detection slightly less robust. For instance, in the case of scaling/unscaling we may select the first 100 components of α from $\mathbb{R}^d$ to $\mathbb{R}^{100}$ and W from $\mathbb{R}^{d\times d'}$ to $\mathbb{R}^{100\times100}$.

Matching. To match an extracted message (made of m chunks of k bits) with a model's identifier, we compute the number s of chunk-wise errors with respect to all possible identifiers. We return a match if s is bellow a fixed threshold τ to ensure resilience to corruption and to provide a confidence score. A theoretical p-value, *i.e.* the probability of obtaining a number of errors lower than s for a random model, is given by the regularized incomplete beta function $\mathcal{I}$:

$$\text{p-value}(s) = 1 - \left(1 - \mathcal{I}_{1/2^k}(m-s, s+1)\right)^N, \qquad (9)$$

where N is the number of distributed models. N accounts for the fact that when multiple models are distributed, the risk of finding a match by chance is higher.

Robustness and security. Watermarking models through invariance is stealthy, because it does not change their outputs and does not leave artifacts in the weights. However, a distortion-free watermark is also a weakness. On the one hand, Alice can hide the watermark without impacting the model's utility, on the other hand, an adversarial Bob may do the same at no cost. In short, most of these watermarks are very robust against classical model manipulations (fine-tuning, quantization, etc.) but not against a malicious user who knows the method. In this case we would only know that the model is an unauthorized copy, without knowing the leaker.

4. EXPERIMENTS

The purpose of the experiments is to evaluate the effectiveness and the robustness of the watermarks to transformations on transformers, in the context of large language models.

4.1. Setup

Model. We use LLaMA [2] models as main benchmark. The architectural differences with regards to the original transformer architecture are pre-normalization [27] with RMSnorm [30], SwiGLU activation [28] and rotary embeddings [31]. Unless stated otherwise, we use the 7B-parameter model. To evaluate that the utility of the model is not degraded, we show results on a next-token prediction task. This is done on random sequences of text taken from Wikipedia, then tokenized using the default tokenizer of LLaMA.

Table 1: Distortion induced on generation (measured as the proportion of generated tokens that differ between the original and the watermarked model); and robustness of watermark extraction under various processes. Each line stands for a different invariant, "all" is the combination of the three invariants in the following order: perm, QK and scaling. We present results of the sped-up extraction, the ones for no speed-up are given as (acc).

Method	Distortion	Byte accuracy (%) on:			
		Noise 1.0	Quant. 3b	Prun. 50%	Fine-tune
Perm.	0.20%	51.4 (99.6)	72.0 (100.0)	100.0	100.0
QK	0.18%	100.0	100.0	100.0	100.0
Scaling	0.24%	100.0	98.1 (100.0)	100.0	100.0
All	1.77%	60.8 (99.8)	70.0 (99.4)	100.0	100.0

Attacks. We consider the following attacks. *Fine-tuning.* We fine-tune the model in a supervised manner with the same settings as [34], on 3 epochs with learning-rate of 2×10^{-5}. *Noise.* We add zero-mean Gaussian noise with standard deviation σ to the model weights. *Quantization.* We quantize the model weights into b bits. To allow flexible rates and ease the experiments, this is done by uniformly quantizing the weights between their minimum and maximum values. *Pruning.* We prune the model weights by zeroing the ones with smallest L1 norms, with sparsity given in percentage of zero weights.

Watermark settings. We apply the encoding process of Sect. 3.2. For permutation invariance, we permute attention heads and FFN layers. For scaling, we alter the layers' RMSnorms and following matrices. The scaling vector α is such that $\log_{10}(\alpha) \sim \mathcal{U}(-1,1)$. For QK products, as mentioned in §3.1, the invertible matrix has to be block diagonal of 2 by 2 rotation matrices, so we randomly sample $d/2$ rotation angles. We fix the number of possible choices at k=8, *i.e.* each choice is encoded with a byte. Therefore, we encode 2 bytes at every layer, except in QK products where we encode 1. When combining all invariants together, we proceed the same way for all blocks: we start with permutation, then apply invertible matrices in QK products, then scale/unscale the layer norms and matrices.

4.2. Results.

Robustness. We evaluate the robustness of the watermark using the *byte accuracy*, *i.e.* the percentage of bytes correctly recovered. Results are averaged over N=100 watermarked models except for fine-tuning where we only fine-tune one model. The model has L=32 layers so the watermark is 64 bytes long except for the QK products invariant where it is 32. In the case of combined invariants, the total number of bytes is 160. We speed-up the extraction by selecting a subset of 100 rows of the matrices (see Sect. 3.2); time needed for extraction is around 20 minutes when using the full matrix instead.

Table 1 reports the byte accuracy for different processing applied before extraction. We observe that the watermark is robust to all attacks with byte accuracy >50%. Errors mainly come from the speed-up of the extraction process. We also consider the *p-value* of the associated statistical test (see Eq. 9). A byte accuracy of 50% on 64-bytes messages is more than enough to reliably identify a model: the p-values are always bellow 10^{-60}, due to the very low probability of simultaneously observing a match between tens of pairs of random bytes. As an illustration, 8 matching bytes on 64-bytes messages already gives a p-value of 10^{-8}.

Table 2: Computational cost of watermark insertion and extraction for different model sizes and the different invariants.

Model	L	d	Insertion (s)			Extraction (s)		
			Perm.	Scaling	QK	Perm.	Scaling	QK
7b	32	4096	3.5	2.7	7.4	9.2	31.7	6.0
13b	40	5120	7.0	4.9	15.8	14.1	30.3	7.7
30b	60	6656	19.3	8.7	47.3	31.7	54.7	13.5
70b	80	8192	37.1	17.5	106	56.3	110	21.5

Model's utility. In fact, previous invariants are not perfect because of quantization (weights are stored as 16bits floating point numbers). Thus, we quantitatively compare watermarked and original models. We feed to both of them 1k sequences of 256 tokens. Predicted next tokens are greedily chosen as the argmax of the 256k observed logits.

Table 1 reports the proportion of predicted tokens that differ between watermarked and original models. As expected this proportion is very low (<1.8%) and higher for the scaling invariant since it further affects quantization. Besides, the distortion increases when the token is far in the sequence *e.g.* for sequences of length 1024 tokens, the average distortion at the last token rises to 2.5% for the scaling invariant. This is still very low and does not affect the utility of the model since predicted tokens are still likely.

Computational efficiency. Larger models have more layers and parameters, which increases the watermark capacity but also the computational cost of insertion and extraction. In Table 2, we report results for different model sizes. Insertion and extraction times are averaged over 100 runs and measured on 2 Intel(R) Xeon(R) 6230 @ 2.10GHz cores and a total of 480GB of RAM. The low computational costs and requirements (no GPU needed) makes it possible to scale to very large models.

5. CONCLUSION

Our work presents a lightweight approach for watermarking large transformers. We leverage invariance properties to generate equivalent copies for watermark embedding. It ensures that the model's outputs are preserved while providing close-to-perfect robustness against processes like fine-tuning or quantization.

Yet, this approach has limitations. Namely, it is limited to white-box scenarios. Additionally, if a sophisticated attacker identifies all invariants, they may remove the watermark by applying the same transformation techniques. In this case, it would still be possible to identify that the model is an unauthorized copy but without the corresponding binary signature. Overall, this work is a starting point to exploit invariance properties that stem from the extreme redundancy of parameters of large networks, for watermarking applications.

6. REFERENCES

[1] OpenAI, "Gpt-4 technical report," *arXiv*, 2023.

[2] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023.

[3] M. AI, "Mistral 7b," *arXiv preprint arXiv:2310.06825*, 2023.

[4] AnthropicAI, "Introducing claude," 2023.

[5] S. Pichai, "An important next step on our AI journey," *Google AI Blog*, 2023.

[6] M. Dehghani, J. Djolonga, B. Mustafa, P. Padlewski, J. Heek, J. Gilmer, A. P. Steiner, M. Caron, R. Geirhos, I. Alabdulmohsin, *et al.*, "Scaling vision transformers to 22 billion parameters," in *International Conference on Machine Learning*, pp. 7480–7512, PMLR, 2023.

[7] M. Oquab, T. Darcet, T. Moutakanni, H. Vo, M. Szafraniec, V. Khalidov, P. Fernandez, D. Haziza, F. Massa, A. El-Nouby, *et al.*, "Dinov2: Learning robust visual features without supervision," *arXiv preprint arXiv:2304.07193*, 2023.

[8] Y. Uchida, Y. Nagai, S. Sakazawa, and S. Satoh, "Embedding watermarks into deep neural networks," in *Proceedings of the 2017 ACM on international conference on multimedia retrieval*, pp. 269–277, 2017.

[9] Y. Adi, C. Baum, M. Cisse, B. Pinkas, and J. Keshet, "Turning your weakness into a strength: Watermarking deep neural networks by backdooring," in *27th USENIX Security Symposium (USENIX Security 18)*, pp. 1615–1631, 2018.

[10] L. Fan, K. W. Ng, C. S. Chan, and Q. Yang, "Deepip: Deep neural network intellectual property protection with passports," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 01, pp. 1–1, 2021.

[11] L. Fan, K. W. Ng, and C. S. Chan, "Rethinking deep neural network ownership verification: Embedding passports to defeat ambiguity attacks," *Advances in neural information processing systems*, vol. 32, 2019.

[12] J. Zhang, D. Chen, J. Liao, W. Zhang, G. Hua, and N. Yu, "Passport-aware normalization for deep model protection," *Advances in Neural Information Processing Systems*, vol. 33, pp. 22619–22628, 2020.

[13] K. Kallas and T. Furon, "Rose: A robust and secure dnn watermarking," in *2022 IEEE International Workshop on Information Forensics and Security (WIFS)*, pp. 1–6, IEEE, 2022.

[14] Y. Yan, X. Pan, M. Zhang, and M. Yang, "Rethinking white-box watermarks on deep learning models under neural structural obfuscation," in *32th USENIX security symposium (USENIX Security 23)*, 2023.

[15] B. Darvish Rouhani, H. Chen, and F. Koushanfar, "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*, pp. 485–497, 2019.

[16] H. Chen, B. D. Rouhani, C. Fu, J. Zhao, and F. Koushanfar, "Deepmarks: A secure fingerprinting framework for digital rights management of deep learning models," in *Proceedings of the 2019 on International Conference on Multimedia Retrieval*, pp. 105–113, 2019.

[17] T. Wang and F. Kerschbaum, "Riga: Covert and robust white-box watermarking of deep neural networks," in *Proceedings of the Web Conference 2021*, pp. 993–1004, 2021.

[18] H. Liu, Z. Weng, and Y. Zhu, "Watermarking deep neural networks with greedy residuals.," in *ICML*, pp. 6978–6988, 2021.

[19] J. Zhang, Z. Gu, J. Jang, H. Wu, M. P. Stoecklin, H. Huang, and I. Molloy, "Protecting intellectual property of deep neural networks with watermarking," in *Proceedings of the 2018 on Asia conference on computer and communications security*, pp. 159–172, 2018.

[20] J. H. Lim, C. S. Chan, K. W. Ng, L. Fan, and Q. Yang, "Protect, show, attend and tell: Empowering image captioning models with ownership protection," *Pattern Recognition*, vol. 122, p. 108285, 2022.

[21] D. S. Ong, C. S. Chan, K. W. Ng, L. Fan, and Q. Yang, "Protecting intellectual property of generative adversarial networks from ambiguity attacks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 3630–3639, 2021.

[22] P. Fernandez, G. Couairon, H. Jégou, M. Douze, and T. Furon, "The stable signature: Rooting watermarks in latent diffusion models," *ICCV*, 2023.

[23] C. Kim, K. Min, M. Patel, S. Cheng, and Y. Yang, "Wouaf: Weight modulation for user attribution and fingerprinting in text-to-image diffusion models," *arXiv preprint arXiv:2306.04744*, 2023.

[24] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[25] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *The Journal of Machine Learning Research*, vol. 21, no. 1, pp. 5485–5551, 2020.

[26] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.

[27] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[28] N. Shazeer, "Glu variants improve transformer," *arXiv preprint arXiv:2002.05202*, 2020.

[29] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[30] B. Zhang and R. Sennrich, "Root mean square layer normalization," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[31] J. Su, Y. Lu, S. Pan, A. Murtadha, B. Wen, and Y. Liu, "Roformer: Enhanced transformer with rotary position embedding," *arXiv preprint arXiv:2104.09864*, 2021.

[32] O. Press, N. Smith, and M. Lewis, "Train short, test long: Attention with linear biases enables input length extrapolation," in *International Conference on Learning Representations*, 2021.

[33] A. Kazemnejad, I. Padhi, K. N. Ramamurthy, P. Das, and S. Reddy, "The impact of positional encoding on length generalization in transformers," *arXiv preprint arXiv:2305.19466*, 2023.

[34] R. Taori, I. Gulrajani, T. Zhang, Y. Dubois, X. Li, C. Guestrin, P. Liang, and T. B. Hashimoto, "Stanford Alpaca: An instruction-following LLaMA model," 2023.