ALMA MATER STUDIORUM
UNIVERSITÀ DI BOLOGNA

Scuola di Scienze

Corso di Laurea in Informatica

Towards the Watermarking of Large Language Models: A Survey

Relatore:
Chiar.mo Prof.
Danilo Montesi

Correlatore:
Chiar.mo Prof.
Flavio Bertini

Presentata da:
Enis Brajevic

Sessione Luglio 2025
Anno Accademico 2024/2025

The happiness of
your life depends upon the
quality of your thoughts

Marcus Aurelius, Meditations

Sommario

Il furto di proprietà intellettuale tramite la condivisione illecita di contenuti testuali è un problema attuale ed urgente, reso ancor più critico dal rapido sviluppo dei Large Language Models (LLM). Il livello di popolarità e di performance raggiunti da questi modelli, progettati per il Natural Language Processing, li rende particolarmente vulnerabili ad abusi, come la generazione di materiale falso o la falsa attribuzione d'autore del testo generato. Il watermarking testuale è il miglior strumento per proteggere gli LLM da questi usi scorretti. Vista la complementarità dei Large Language Models e del watermarking testuale, in questo studio descriviamo lo stato dell'arte di questi due campi e il modo in cui possono essere combinati. Il nostro contributo può essere riassunto nei seguenti quattro punti principali: (1) una panoramica delle caratteristiche fondamentali degli LLM e del loro stato dell'arte; (2) una panoramica e un'analisi dei principali tipi di watermarking testuale applicati al testo esistente, e un confronto tra di essi; (3) una panoramica e un'analisi dei principali metodi di watermarking applicati agli LLM; (4) l'identificazione di aree poco esplorate con la formulazione di domande di ricerca mirate, a cui viene data risposta successivamente allo studio su LLM e watermarking testuale.

Abstract

Intellectual property theft through the unlawful sharing of texts is a pressing issue, further intensified by the rapid development of Large Language Models (LLMs). These models, designed for Natural Language Processing tasks, have reached a level of performance and popularity that makes them very vulnerable to misuse, such as the generation of fake text or false authorship attribution of the generated content. Text watermarking is the best tool to safeguard LLMs from such misuse. Because Large Language Models and text watermarking exhibit such complementarity, in this comprehensive survey we describe the state of the art of these two fields and how they can be combined. Our contribution can be summarized in the following four main aspects: (1) an overview of the fundamental characteristics of LLMs and their state of the art; (2) an overview and analysis of the main types of text watermarking methods that target existing text, and how they compare to each other; (3) an overview and analysis of the main LLM watermarking methods; (4) identification of underexplored areas through the formulation of targeted research questions, which are then addressed after the comprehensive study on LLMs and text watermarking.

Acknowledgements

In addition to my supervisors Prof. Danilo Montesi and Prof. Flavio Bertini, I would like to express my sincere gratitude to Dott. Simone Branchetti, a very insightful PhD student who gave me valuable advice and constant support throughout the work on my thesis.

Contents

Chapter 1

Introduction

Intellectual property theft through the unlawful sharing of texts is an ongoing problem in current times.
This problem is exacerbated by the rapid development of Large Language Models, deep learning models specifically designed for Natural Language Processing tasks. LLMs have seen a rise in performance and popularity in recent years [1][2], and we have reached a point where their text generation capabilities make it so that LLMs pass the Turing Test [3], leading inevitably to dangerous scenarios. For instance, it has been shown that LLMs trained on news articles can generate realistic fake news indistinguishable from human-written news to the human eye [4].
Furthermore, LLMs are often trained on datasets containing intellectual property whose usage has not been authorized by the rights holders.
Because of this, the importance of tracing text to its owner and discerning between machine-generated text and human-authored text has become paramount.
Here, we explore how text watermarking and LLMs complement each other effectively in addressing these challenges, contributing to the literature by:

- Organizing watermarking techniques proposed by the literature and bringing clarity to the matter.
- Highlighting what methods can be used to solve our research questions regarding parts of the literature that remain underexplored.

1.1 What is Watermarking?

Watermarking is the marking of a resource (such as an image, video, or text) by embedding digital information in it that can later be extracted. It is a well-recognized technique used to identify the ownership of a resource and protect intellectual prop-

erty, and it is, in fact, regarded as the most suitable application for this matter [5]. Digital watermarking has been applied mainly to images for provenance tracing, with the first techniques dating back to the 1990s [6].
Watermarking is part of the digital information hiding techniques (any technique that involves concealing information within other data or media). It is a common error to confuse watermarking with steganography, another method within the same domain that serves a different purpose: watermarking is used to embed digital information with the goal of reaching copyright protection, with the digital information that may be known to exist and also visible (though, of course, this is undesirable), while steganography is a means of secret communication that aims to conceal the very existence of a hidden message, enabling covert communication [7].

1.2 Why Text Watermarking?

Text is one of the most prevalent types of resources on the web and, as such, requires a high level of protection. Intellectual property can be misappropriated from academic papers, while news can be tampered with to generate fake news that can be widely disseminated through social media platforms and online news articles.
Text watermarking is much needed to solve these issues: by extracting the watermark containing information on the original owner and text, problems like intellectual property misappropriation and fake news dissemination can be dismantled. Contrary to image watermarking, a field where great results have been obtained due to the noise-resistant nature of images, that are able to bear digital information efficiently without it being noticeable to the eyes of a human observer, text watermarking conceals some intrinsic complications: text is not noise-resistant like images, leading to the uneasy task of finding a good balance to embed a watermark that is both robust and imperceptible to the reader.
More complications stem from the fact that semantic and syntactic rules of the specific language need to be leveraged to embed the watermark and, furthermore, excerpts of text could be extracted leading to watermark deletion (cropping is also very common with images, but most of the content is kept).
Ultimately, substantial progress has been made in the field of text watermarking in the past few years, but much more research remains to be done to satisfy the needs, especially with the rapid development of LLMs. A comprehensive overview, such as the one presented here, serves as a valuable resource for understanding the state of the art and identifying areas where further contributions are needed.

1.3 Why Large Language Models?

The recent advancements in the performance of Large Language Models have significantly boosted their relevance and usage, so that, for example, ChatGPT has become the fifth most visited website in the world [1] and it has set the record for fastest-growing user base [2]. This surge in usage and performance of LLMs leads to an increased usefulness of text watermarking, with new important use cases benefiting both:

- **LLMs enhance text watermarking techniques**
  LLMs (and other deep models) can be used to embed watermarks in texts and build sophisticated watermarking techniques that avoid rigid traditional rule-based methods [8][9].
  See Section 3.4 for a more detailed explanation of how LLMs (and other deep models) enhance text watermarking techniques.

- **Watermarking of generated text to prevent misuse**
  A watermark could be applied to the text generated by an LLM to prevent misuse such as false authorship (where the user could pose as the writer of that text) or the spreading of fake information [9][10][11]. Detecting this watermark would mean that the text was generated by an LLM, hence unmasking attempts at misuse.
  See Section 4.1.2 for a more detailed explanation of how watermarking can be employed to prevent misuse.

- **Dataset protection**
  LLMs are trained on huge datasets, which contain text data (such as papers, news articles, or code) that could be used without the consent of the rightful owners.
  Watermarking can be used to mark sources and later detect if they were used in the training process of an LLM [12][13][14]. See Section 4.1.3 for a more detailed explanation of what is dataset protection, and how text watermarking helps solve this problem.

1.4 Research questions

In addition to giving an overview of what has been done in the field of text watermarking and Large Language Models, we contribute by answering research questions regarding parts of the literature that remain underexplored and would benefit from further investigation:

1. **Given a text generated by a Large Language Model and a set of LLMs, is it possible to determine if one of the LLMs in the set has generated that text?** (see Figure 1.1)

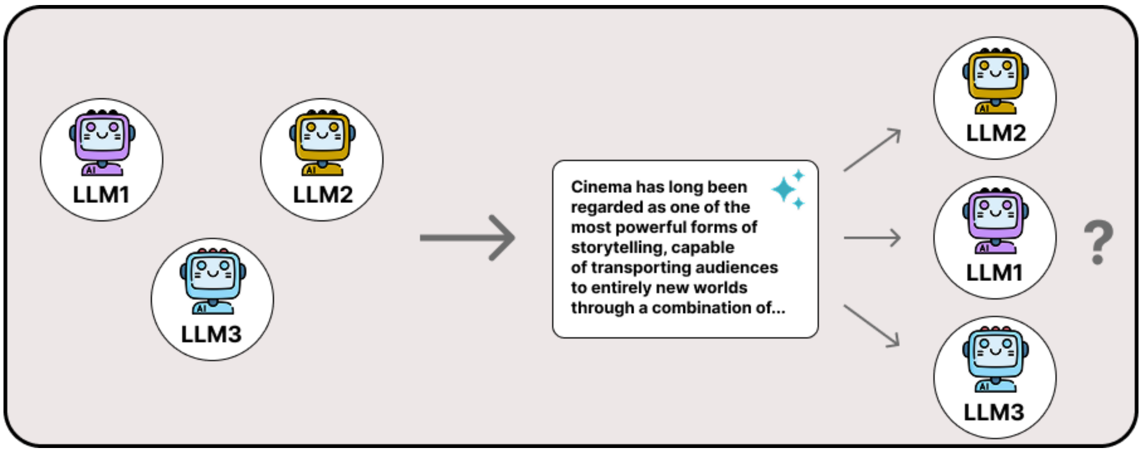
LLM1
LLM2
LLM3
Cinema has long been regarded as one of the most powerful forms of storytelling, capable of transporting audiences to entirely new worlds through a combination of...
LLM2
LLM1
LLM3
?

Figure 1.1: If we have a set of LLMs containing LLM1, LLM2 and LLM3, and a text generated by an LLM is given to us, is it possible to determine if one of the LLMs in the set has generated that text?

The next step from identifying if a text was generated by an LLM, is identifying which LLM generated that text, given a set of candidates. This problem is typically referred to as *authorship attribution*; here, it is addressed in the context of Large Language Models. If it were possible to determine which LLM generated a certain text, we could protect the intellectual property of the specific LLM.

2. **Given a text generated by a Large Language Model, is it possible to determine if it was generated from watermarked documents?** (see Figure 1.2)

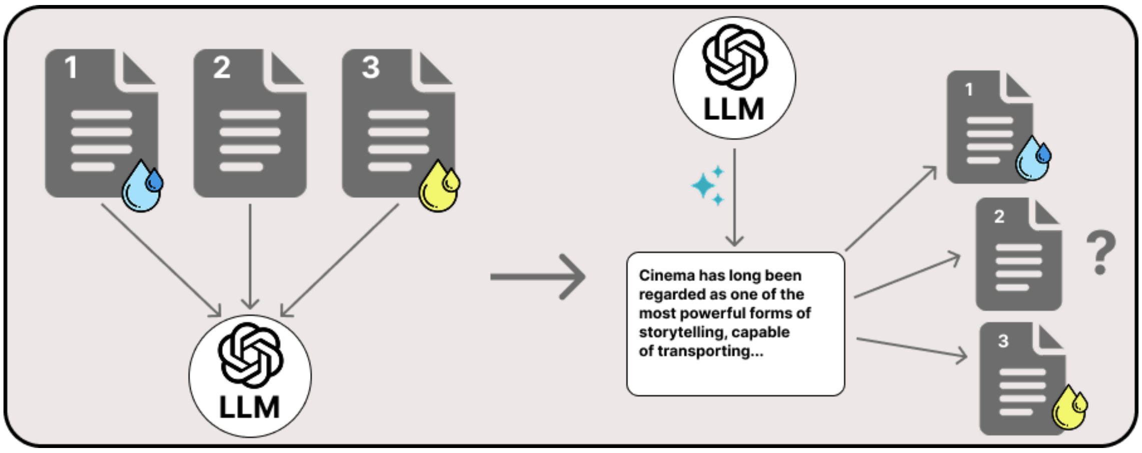
1
2
3
LLM
LLM
Cinema has long been regarded as one of the most powerful forms of storytelling, capable of transporting...
1
2
3
?

Figure 1.2: If the LLM is trained on watermarked and non-watermarked documents, is it possible to determine if a watermarked document has been used to generate a given text?

Determining if a text generated by an LLM was generated from specific watermarked documents would be a useful tool to protect a text document from being used in the training process of an LLM without the consent of its owners. Solving this research question would, in fact, make it possible to solve research question 3, since it would be possible to determine that the LLM was trained on a given text document.

3. **Given a text document, is it possible to detect if a Large Language Model has been trained on that document?** (see Figure 1.3)

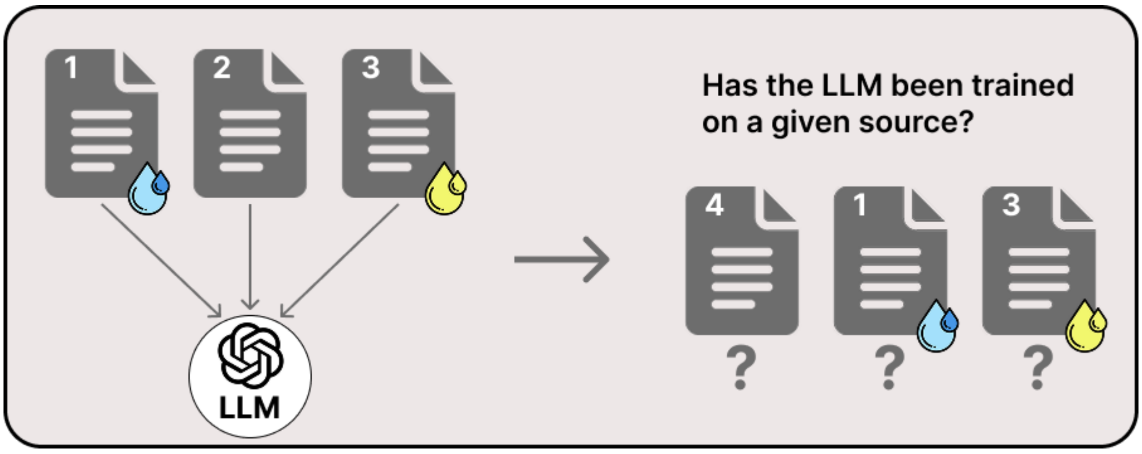

Figure 1.3: If the LLM is trained on watermarked and non-watermarked documents, is it possible to determine if a given source has been used in the training process of the LLM?

Answering this question would give us the chance to verify if an LLM has been trained on a text source for which the consent was not given by the rightful owners. In this way, it is possible to protect the text data on which the LLM is trained.

All research questions have been answered in Section 5, after a thorough examination of the state of the art of LLMs and text watermarking.

Chapter 2

Large Language Models

2.1 What are Large Language Models?

Natural Language Processing (NLP) is a branch of Artificial Intelligence that aims to provide computers with the ability to work with natural language data.
Large Language Models (LLMs) are deep learning models specifically designed for NLP tasks. They are high-performing text generators, but beyond that, LLMs are capable of performing many more tasks: Machine Translation [15] (translate text from one language to another), Information Retrieval [16] (find relevant documents or information from a large collection, in response to a user query), Text Summarization [17] (summarize long pieces of text into shorter versions while retaining the original meaning), Sentiment Analysis [18] (determine the sentiment expressed in a piece of text), Question Answering [19] (answer specific questions based on a given context or document, or even general knowledge), Text Classification [20] (classify text into predefined categories based on its content) and many more.
They have gathered much attention in recent years; in fact, due to their development, they have revolutionized the field of NLP, making it possible for users to enjoy functionalities that were never available before.
For instance, when OpenAI released ChatGPT in 2022 (only three years ago!), the public was captivated by it: that kind of performance had never been reached before, making it so that people were both enthusiastic about all the useful things it could do and frightened of what AI could achieve in the future.

2.2 Transformers

The Transformer [21] is the architecture that revolutionized NLP and that made it possible for LLMs to achieve great results.
Before their introduction, LLMs were primarily based on Recurrent Neural Networks such as LSTMs [22]: these models process sequences one token at a time, using for each token a hidden state updated at each step, that acts as a memory, capturing information from all earlier tokens in the sequence. This recurrent design is what gives RNNs their power, but the fact that it is sequential means you cannot compute the hidden states in parallel and, furthermore, long-range dependencies between words tend to fade away over time.
The attention mechanism, first introduced by [23], was then used to allow RNN-based models to focus on the relevant parts of the input sequence when generating outputs. Instead of treating all parts of the input sequence equally, attention mechanisms assign varying importance to different elements, enabling the model to focus more on the most relevant information. The introduction of attention mechanisms allowed models to better understand the dependencies between words, but still, due to the increased use of deep learning in NLP, there was a growing demand for architectures that could scale and handle long sequences, while also implementing attention.
The Transformer architecture was then introduced in 2017 [21] to address this issue.

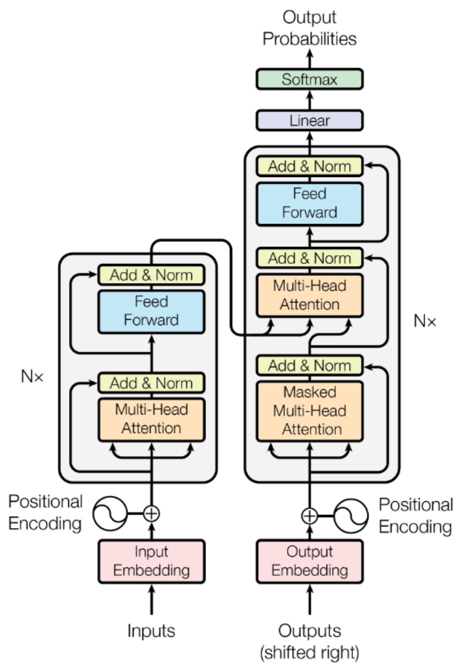

Figure 2.1: The Transformer encoder-decoder architecture, from [21]

Transformers overcame the limitations of traditional attention mechanisms by eliminating recurrence and introducing self-attention. Unlike RNNs, Transformers process all words at the same time, avoiding sequential processing and enabling parallel processing across input sequences [24].
Each word can look at other words in the sequence using the new self-attention mechanism. This involves computing a vector of attention scores, with values between 0 and 1, which represents how much focus the model should place on each of the other words when processing the current word.
To capture different types of relationship between words, Transformers use multi-head attention, which, for each token, runs multiple self-attention mechanisms in parallel. Each head learns to focus on different aspects of the sequence, such as syntactic structure or semantic meaning, providing richer contextual understanding. In particular, each head calculates attention scores and uses these scores to compute an output vector. The vectors generated from each head are concatenated into a longer vector that is then passed through a linear projection layer, producing a final vector for that token that contains contextualized information (knows about other tokens in the sequence).
Since all state-of-the-art LLMs are based on the Transformer architecture, we are now going into the details of its encoder-decoder structure (see Figure 2.1).

2.2.1 Token embeddings

Tokens are textual units, such as words or sub-words, that constitute the training data for Machine Learning (ML) models. Since ML models are designed to process numerical data, textual inputs must first be converted to a numerical format. The first step to achieve this is tokenization, which breaks text down into tokens [25]. Following tokenization, each token is mapped to a unique integer and from that to a continuous vector space. The representations of tokens in the vector space are called token embeddings. These embeddings allow the ML model to understand word meanings, and they are learned from data so that they capture semantic relationships (for example, the vectors for "happy" and "joyful" should be close together in the embedding space, as they have a similar meaning).

2.2.2 Transformer Encoder

The encoder takes as input the input sequence embeddings, together with positional encoding, which provides information about the positions of tokens. The encoder uses multi-head attention to capture context and a feedforward network to improve representation and better capture intricate patterns within the sequence. The encoder outputs transformed representations of the input tokens, now contextualized.

2.2.3 Transformer Decoder

The decoder takes the target sequence (shifted right) embeddings combined with positional encoding, and its goal is to correctly predict the output at each position, given the previously generated tokens.
To achieve this goal, the decoder does masked self-attention over target tokens (each token cannot see future tokens) and cross-attention over the output of the encoder to incorporate contextual information from the input sequence. The result then passes through a feedforward network, generating a sequence of vectors, one for each token position in the output. Each vector represents the model's internal, context-enriched representation of that token, incorporating both the token's identity and its relationship to other tokens in the sequence. All of these vectors are then passed in parallel through a linear layer, projecting each of them into a vector the size of the vocabulary, called logits. Finally, the logits are passed through a softmax to generate a probability distribution over the vocabulary for each token position, with the goal of minimizing the loss between the predicted probability distribution over the vocabulary and the actual next token.

As just explained, thanks to attention, the Transformer architecture is particularly good at capturing context in sequential information and modeling how tokens in a sequence relate to each other. This, together with the fact that the architecture enables parallelization, is why Transformers have replaced RNNs to be the state of the art for LLMs.

2.3 Types of LLM architectures

Due to the efficient parallelization and context-capturing abilities of Transformers (as explained in Section 2.2), they have become the backbone of all state-of-the-art LLMs [26].
Although all modern LLMs are built upon the Transformer architecture, there are slight variations among them. We classify LLMs into four types, based on their architecture: encoder-decoder, encoder-only, decoder-only, and hybrid [27].

2.3.1 Encoder-decoder

The vanilla Transformer model [21] is based on the encoder-decoder architecture.
Models with this architecture use the encoder to process the input data and compress the information into a context-rich representation, and the decoder to generate the output sequence from this representation (for more details, see Section 2.2). So far, only a small number of LLMs are based on the encoder-decoder architecture, an

example is Flan-T5 [28]. This type of architecture is used for translation or other text-to-text tasks.

2.3.2 Decoder-only

Decoder-only models use only the Transformer decoder in their architecture, with input and output tokens processed in the same way by the decoder [26].
The decoder-only architecture is said to be causal because it follows the causality of language: each processed token attends only to previous tokens and itself, not to future tokens. This makes decoder-only models particularly well-suited for autoregressive tasks like text generation.
Most of the state-of-the-art language models today adopt a decoder-only architecture, particularly those designed for text generation. A prime example is the GPT family, consisting of language models (one of the latest versions being GPT-4 [29]) built on a causal decoder-only Transformer architecture, and whose recent models have achieved great performance.

2.3.3 Encoder-only

Encoder-only models use a different modeling technique: their approach is to mask a token in the sequence and try to predict it while considering the surrounding context [9]. This method of training is commonly known as Masked Language Modeling (MLM). For example, the sentence *Rome is the capital of Italy* can be transformed in *Rome is the [MASK] of Italy.* The [MASK] token replaces a token in the sentence and the model is trained to predict the correct word, looking at the whole sentence.
Encoder-only models are said to be non-causal, since in contrast to decoder-only models, they allow all input tokens to attend to each other in the attention mechanism, ignoring the causal structure of language. This means that while the attention mechanism in decoder-only models is unidirectional (only previous tokens), here the attention mechanism is bidirectional (allowed to look at both previous and future tokens). Non-causal models tend to perform better in natural language understanding rather than text generation, with bidirectional attention making them powerful at tasks like question answering. The main examples of this category of models are BERT [30] and RoBERTa [31].

BERT

BERT [30] is trained using two unsupervised tasks: Masked Language Modeling (input tokens are masked at random and then a prediction of those tokens is made) and Next Sentence Prediction (given two portions of text A and B, it is predicted if sentence B is the successor of sentence A).

The input embeddings are obtained (see Figure 2.2) by pairing two sentences in a single sequence and adding the special start token [CLS] (placed at the beginning of the sequence) and separator token [SEP] (placed between two sentences). In particular, the input embeddings are obtained by summing together the token embeddings with segment embeddings (indicate if the token pertains to sentence A or sentence B) and position embeddings (capture the position of the token in the sentence).

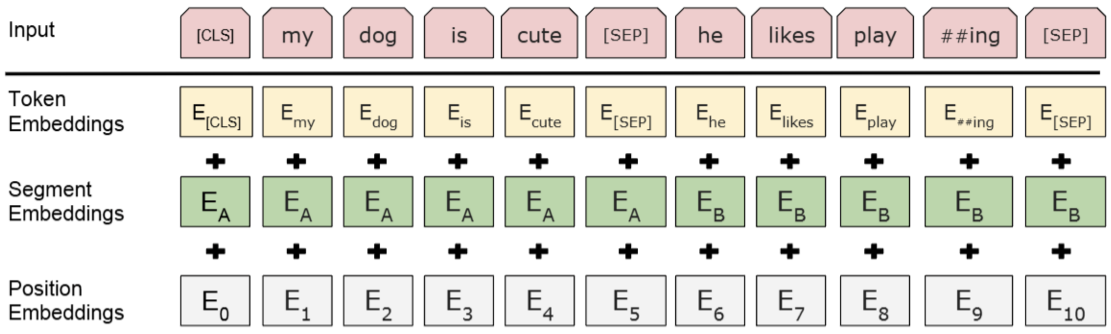

Figure 2.2: BERT model input sequence, from [30]

From these input embeddings, BERT is trained using Masked Language Modeling and Next Sentence Prediction objectives.

RoBERTa

RoBERTa [31] is a variant of BERT that improves the performance of BERT by modifying the training strategy and not the model architecture.
It does so by using dynamic masking: instead of generating and saving only one masked version of each sentence, RoBERTa dynamically creates different masked versions during training, allowing the model to learn from multiple masking patterns for the same input.
Furthermore, RoBERTa removed the NSP training task because it was deemed unhelpful and in certain cases even harmful, and augmented training details such as size of batches, dataset size and training passes.
These changes allow RoBERTa to perform better than BERT.

2.3.4 Hybrid

In addition to the previous model architectures, hybrid models also exist.
These models integrate aspects of both causal and non-causal models by dividing the input text in two parts. The first part mirrors the non-causal language modeling

(bidirectional), while the second part mirrors the causal language modeling (unidirectional).
Multiple popular models such as T5 [32] and BART [33] belong to this category.

2.4 Pre-training and fine-tuning

Pre-training is the first stage in building an LLM, in which the model learns general patterns from vast amounts of unlabeled data prior to fine-tuning on specific tasks [26]. We now describe the multi-stage pipeline that plays a crucial role in shaping the LLM's capabilities.

2.4.1 Data preparation

Compared to smaller language models, LLMs require higher-quality data for effective pre-training, and their performance heavily depends on both the size and the preprocessing of the pre-training corpus.
The data preparation process can be summarized in three key steps [26]:

1. **Data collection**
   To build an effective LLM, it is essential to gather a substantial amount of natural language data from a wide range of sources. Most existing LLMs rely on a combination of diverse public textual datasets for pre-training. In particular, the pre-training corpus can generally be divided into two categories: general-purpose data and domain-specific data. General-purpose data such as web pages, books, and conversational text is widely used by LLMs due to its abundance, diversity, and accessibility, all of which contribute to enhanced language modeling and generalization. Recognizing the strong generalization abilities of LLMs, some research efforts have expanded the pre-training corpus to include domain-specific datasets, such as multilingual content, scientific papers and code. This allows LLMs to develop more specialized task-solving capabilities.

2. **Data cleaning**
   After data collection, data cleaning, with the goal of augmenting the quality of data, is key to pre-train an LLM. In this phase, de-duplication of data is usually done at different granularities, including sentence-level, document-level, and dataset-level. Furthermore, low-quality text, harmful content (such as hate speech), and data with privacy concerns is removed. Finally, tokenization is done to segment raw text into sequences of individual tokens, which are subsequently used as the inputs of LLMs.

3. **Data scheduling**
   After data cleaning, it is crucial to develop effective strategies for organizing the use of multi-source data during the pre-training of a capable LLM. Two key aspects in this are the proportion of each data source and the order in which each data source is scheduled for training. The first aspect aims at increasing the diversity of data sources, while also increasing the proportion of data sources that could provide positive influences on the performance of the downstream task, or in general, enhance certain model abilities. The second key aspect has the goal to schedule correctly the order in which specific data is presented to LLMs for pre-training, since to learn a target skill, the order in which basic skills are learned is important.

2.4.2 Pre-training

After the data preparation process, pre-training can finally be done. In this phase, the data obtained from the data preparation process is used to embed general knowledge into the model's parameters.
The language modeling (LM) objective is the most widely used approach for pre-training decoder-only LLMs [26]. Given a sequence of tokens $x = \{x_1, \ldots, x_n\}$, the goal of the language modeling approach is to auto-regressively predict the target tokens x_i based on the preceding tokens $x_{<i}$ in a sequence. A general training goal is to maximize the following likelihood:

$$\mathrm{L}_{\mathrm{LM}}(x) = \sum_{i=1}^{n} \log P(x_i \mid x_{<i})$$

2.4.3 Fine-tuning

Fine-tuning is performed when deploying a model for downstream tasks. To obtain a fine-tuned model, a pre-trained model is taken and trained on a smaller domain-specific dataset. After doing so, the model retains the general language knowledge obtained from the pre-training but also performs better on the new task.
In this phase, the availability of annotated data plays a big role. Three primary scenarios are possible, with fine-tuning being applicable or not based on the quantity of annotated domain-specific data available [27]:

- **Zero annotated data:**
  In situations where annotated data is lacking, LLMs have proven to be the most effective approach, demonstrating their ability to surpass other zero-shot methods [34].

- **Few annotated data:**
  In cases where there is few annotated data available, a few-shot learning approach is applied by embedding examples directly into the input prompt of the LLMs. These examples help the model generalize to the task at hand.

- **Abundant annotated data:** When a large amount of annotated data is available for a specific task, both fine-tuned models and LLMs are viable options. In many cases, fine-tuning a model can result in a strong fit to the data.

2.4.4 Pre-trained model versus fine-tuned model

Choosing between using a pre-trained model or a fine-tuned model depends on data availability and on the task at hand.
Regarding the first aspect, as explained in Section 2.4.3, a fine-tuning approach is to be pursued only in the case of abundant available data.
On the other hand, a pre-trained model can be preferred to a fine-tuned model based on the target task: pre-trained models usually perform better at general knowledge tasks (such as closed-book question-answering or miscellaneous text classification), while fine-tuned models inevitably perform better at specific tasks like Machine Reading Comprehension (answering questions based on given paragraphs).

Chapter 3

Text Watermarking

Text watermarking is the marking of text data to embed digital information that can later be extracted [5].
Text is the most present type of data on the internet, and text watermarking, as explained in Section 1.2, requires yet much research.
That is why, in this chapter, we provide an overview of text watermarking and the techniques proposed in the literature, aiming to clarify the state of the art in this field.

3.1 Watermarking formulas

To understand what text watermarking is, we start by first explaining its general formulas. To obtain the watermarked text X' from X, a watermark W (generally identified as a sequence of bits) is embedded in X following a specific embedding process, and possibly also using a key K, that is not present in all watermarking methods but is present in some sophisticated ones (see Figure 3.1).

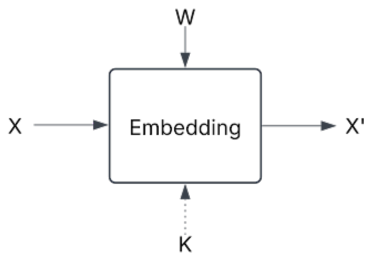

Figure 3.1: Watermark embedding

After the digital data has been embedded, it needs to be extractable, else there would be no use in embedding it.
There are three types of extraction methods: blind, semi-blind and non-blind [5].

In blind watermarking, the extraction is possible without the presence of the original text X nor the watermark W. This is the most favorable case, since, simply, the watermark bits W' can be extracted from the watermarked data X', eventually using the key K if included in the watermarking method (see Figure 3.2).

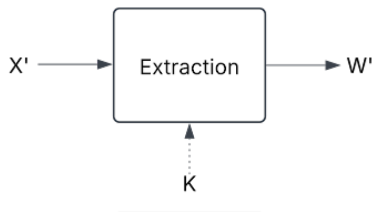

Figure 3.2: Blind watermarking

In semi-blind watermarking, the original watermark W is also needed, in addition to the watermarked text X' and possibly the key K (see Figure 3.3)

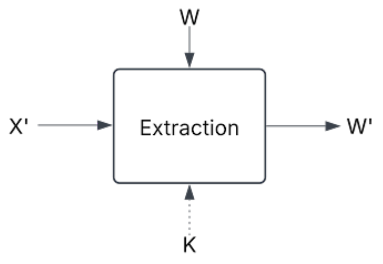

Figure 3.3: Semi-blind watermarking

In non-blind watermarking, we also need the original text X, in addition to the watermark W, the watermarked text X' and possibly the key K (see Figure 3.4). This is the least preferable scenario, since requiring the original text significantly limits the situations in which watermark detection can be performed, given that the original unmarked content is very often not available at the point of verification.

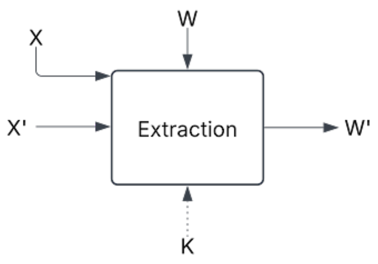

Figure 3.4: Non-blind watermarking

3.2 Properties of watermarking methods

A watermarking method is characterized by multiple properties [5]:

- **Robustness**
  The robustness of a watermarking method is its ability to resist any attack that aims at deleting or modifying the hidden watermark. This property is very important since after embedding the watermark and obtaining a watermarked text, we would like the embedded data to be retrieved in order to verify the ownership of the text. Given that the attacker would like instead to make the ownership of the text data unverifiable, we need to expect that he would do anything possible to change the text and keep the semantics the same, while at the same time removing the watermark.
  There is no precise method that directly measures robustness. However, robustness can be assessed by evaluating how well the watermarked text withstands various attacks aimed at removing the watermark.

- **Imperceptibility**
  The imperceptibility of a watermarking method is its ability to remain unnoticed to a reader. Ideally, we want the watermark to be imperceptible, so that it does not alter the readability, style, or semantics of the original text. This ensures that the watermark remains undetectable to human readers, not hurting the quality of the text, and also to potential attackers seeking weaknesses to exploit to remove the watermark.

- **Blindness**
  A watermarking method can be blind, semi-blind or non-blind, based on what is needed during the extraction process to retrieve the watermark (see Section 3.1). A blind watermarking is the preferable scenario, since it allows to verify the watermark without requiring access to the original text, making the method more practical, scalable, and suitable for real-world applications where

the original content may not be available

- **Capacity**
  The capacity of a watermarking method is the ratio of watermarked data to total data. In other words, how many bits or symbols of watermark can be embedded in a given text. The capacity C can be calculated using the following equation:
  $$C = \frac{N_w}{N}$$
  Where:
  - N_w = number of watermark bits embedded
  - N = total number of bits of the host text

  Generally, different watermarking methods may require different watermarking capacities. It is preferable to use a watermarking method with a high embedding capacity, making it possible to target also short texts.

Given these properties, we can describe the perfect watermarking method as being robust, imperceptible, blind, and also having a high embedding capacity. No watermarking technique that satisfactorily meets these conditions has yet been found, leaving still much space for research.

3.3 Taxonomy of text watermarking methods

Having now understood the basis of text watermarking, with its theoretical formulas and properties, we provide a taxonomy of current watermarking methods, describing the state of the art in this field.
Recently, LLMs have come in the picture, and apart from watermarking existing text, techniques have been developed to modify LLMs to watermark them and the text they generate. Hence, we can differentiate watermarking techniques between watermarking of existing text and watermarking Large Language Models.

Before going into the details of each type, we summarize the possible types of text watermarking methods [35]:

- Watermarking existing text
  - Format based watermarking (see Section 3.3.1)
  - Lexical based watermarking (see Section 3.3.1)
  - Syntactic based watermarking (see Section 3.3.1)
  - Generation based watermarking (see Section 3.3.1)

- Watermarking Large Language Models
  - Watermarking during logits generation (see Section 3.3.2)
  - Watermarking during sampling (see Section 3.3.2)
  - Watermarking during LLM training (see Section 3.3.2)

3.3.1 Watermarking existing text

Watermarking existing text is the standard scenario in which we have a text we would like to protect, and a watermark is embedded into it. The watermarking of existing text can be classified into four types: lexical based, syntactic based, format based, and generation based [35]. Text watermarking methods can be categorized into these four types based on the manner in which the embedding process is done (see Figure 3.5).

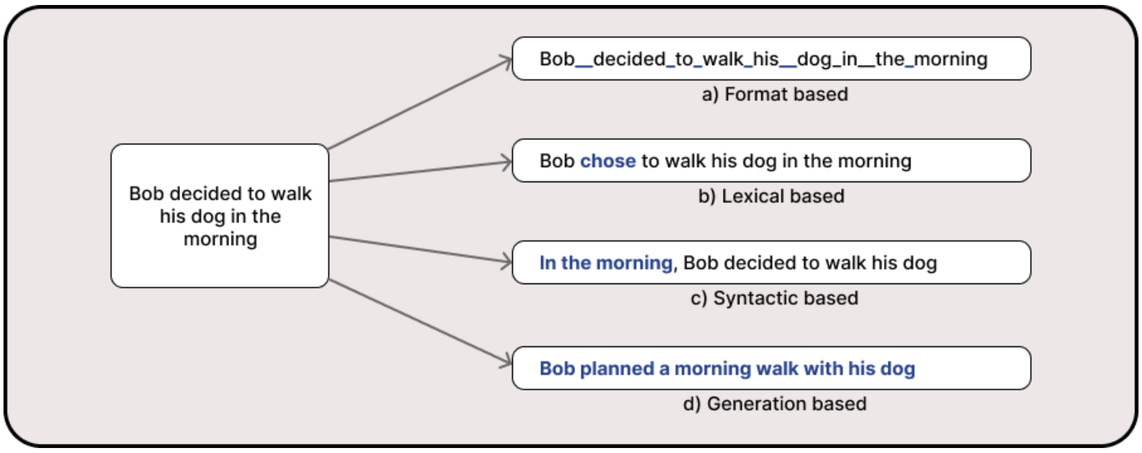
Bob decided to walk his dog in the morning
Bob__decided_to_walk_his__dog_in__the_morning
a) Format based
Bob chose to walk his dog in the morning
b) Lexical based
In the morning, Bob decided to walk his dog
c) Syntactic based
Bob planned a morning walk with his dog
d) Generation based

Figure 3.5: The possible types of watermarking methods for existing text

Format based watermarking

Format based methods refer to techniques that modify only the structure of the text, without changing the actual content (see example *a* of Figure 3.5).

Line/word-shift coding One of the first examples of format based watermarking, dating back to 1995, is the approach of Brassil et al. [6], that encompasses techniques such as line-shift and word-shift to embed the watermark in the text. The line-shift technique alters a document by vertically adjusting the positions of text lines to create a unique encoding of the content based on the watermark message. The word-shift, instead, alters a document by horizontally shifting the locations of words

in the text. The detection process locates shifts by analyzing the spacing between text line profiles or word column profiles. This method aims to embed a watermark without changing the content of the text, and by making the embedding of the watermark almost imperceptible to the user. However, it has a big disadvantage: it applies only to text rendered as images and does not insert a watermark within the text itself.
To address this issue, watermarking methods that leverage the fact that there are multiple Unicode characters representing a blank space have been developed.

WhiteSteg One example of these methods is WhiteSteg [36], that embeds the watermark by combining inter-word and inter-paragraph spacing. A single whitespace character represents a ‘0’, while two whitespace characters represent a ‘1’.

UniSpach Another method that leverages white space encodings is UniSpach [37]. In this method, normal space characters are replaced with a combination of shorter and longer space encodings. Based on the next bits of the watermark to embed, a combination of spaces is used. For example, given eight possible combinations of white spaces, $log_2 8 = 3$ bits can be embedded based on the chosen combination.

Homoglyph Substitution Instead of working on white space encodings, Rizzo et al. [38] introduced a Unicode homoglyph substitution method, using almost identical symbols that are encoded differently to embed a watermark in the text. If a character admits a homoglyph, the bit ’1’ is embedded in the text by substituting the character with its homoglyph. Otherwise, the bit ’0’ is embedded by not applying the substitution. For example, the symbol ’c’, encoded by ’0x0063’ in Unicode, has a homoglyph since the encoding ’0x217d’ reproduces an almost identical symbol. This method embeds a watermark that is much more imperceptible to the reader compared to previous methods like WhiteSteg and UniSpach, since it does not stretch the text much because of additional white spaces. Additionally, it offers greater capacity by leveraging the encodings of all characters, rather than relying solely on white space.

Having now seen some examples of format based watermarking, it is important to note that all of the methods pertaining to this category are susceptible to detection, as they alter the text’s surface formatting. Furthermore, a big disadvantage of these methods is the fact that all of them are weak against rewriting attacks: by simply rewriting the watermarked text, the attacker can successfully remove the watermark and obtain the original unwatermarked text.

Lexical based watermarking

To address the issues of format based watermarking methods, multiple studies have developed techniques that apply word-level modifications. In particular, lexical based watermarking embeds watermarks by replacing words with their synonyms, without changing the sentence's syntactic structure (see example *b* of Figure 3.5).

Equimark An example of a lexical based watermarking method is Equimark, introduced by Topkara et al. [39]. This method embeds a watermark in the text by substituting words with synonyms selected from WordNet [40], an electronic dictionary that organizes English words into synonym sets, each representing one underlying lexical concept, a sense. For example, the word "pitch" has multiple senses: the playing field in games like football, the perceived frequency of a sound, or a persuasive marketing presentation. Topkara's Equimark replaces words with synonyms that have the most number of senses, with the goal of increasing ambiguity to the point where an attacker would find difficulty in automatically removing the watermark and human intervention would be needed.

DeepTextMark A more sophisticated method that employs synonym substitution is DeepTextMark [41]. This method aims at an automatic watermark embedding and detection, without any need of human intervention compared to Equimark, where linguists are needed to produce a dictionary of synonyms. In DeepTextMark, each word candidate for substitution is transformed into an embedding vector using a pre-trained Word2Vec model [42]. A list of substitute words is produced by identifying the n nearest vectors to the embedding of the word in the vector space, and reconverting these vectors back into words. After generating a list of sentence proposals by replacing each candidate word with its corresponding set of alternative words, a pretrained Universal Sentence Encoder is used to produce embeddings for the sentence proposal and for the input sequence. The sentence proposal whose embeddings have the highest similarity score to the input embeddings is selected as the watermarked sentence, after a grammatical adjustment that could be needed since reconversion from the vector space does not always produce a grammatically correct sentence. To detect the watermark, DeepTextMark uses a pre-trained BERT model to generate embeddings and a transformer block to detect if the text is watermarked.

CALS It is important to note the disadvantages of previous methods. In addition to the fact that Equimark depends on a handcrafted lexical vocabulary, that requires costly human work, both Equimark and DeepTextMark employ lexical substitutions that do not take into consideration the context of the substituted word. To address this issue, Yang et al. introduced context-aware lexical substitution (CALS) [43].

The method uses BERT to suggest context-aware substitution candidates by inferring the semantic relatedness between the watermarked sentence and the original sentence.

IF Building on CALS' approach, Yoo et al. introduced a more robust watermarking method, through invariant features: IF[44]. In this approach, words to be substituted are chosen based on a function of the invariant features of the text. The invariant features are words that remain unchanged under utility-preserving corruptions, as altering them would require modifying a substantial portion of the text, and the attacker would have no use in modifying the text to remove the watermark if it would mean losing the meaning carried by the text.

Syntactic based watermarking

Since lexical based watermarking might be weak against attacks like random synonym replacement, methods like the syntactic based ones have been invented. Syntactic based watermarking techniques embed watermarks by changing the syntactic structure of the text (see example *c* of Figure 3.5).
The common step in syntactic based watermarking is building the syntax tree of a sentence, after which some syntactic operations are applied.

Adjunct movement, Clefting, Passivization Atallah et al. [45] introduced three common syntax transformations for embedding watermark messages: adjunct movement, clefting, and passivization. Adjunct movement is when an adjunct (a phrase that adds additional information like time, space, or manner) is moved from its original position. For instance, the sentence *You left after the concert* can be transformed in *After the concert, you left.* Clefting is the process which transforms a sentence into a more complex one. For example, *John ate the cake* can be transformed into *It was the cake that John ate.* Passivization, instead, transforms a sentence from an active to its passive form.

Activization Adding to the previous techniques, Topkara et al. [46] introduced more syntax transformations, such as activization, that transforms a sentence from a passive to its active form.

It is important to signal the disadvantages of this type of watermarking: the watermarking capacity is quite low, since a big chunk of text is needed to change its syntactic structure and embed a long watermark. Furthermore, syntactic methods change the content of the text, leading to a watermarked version that does not always have the same meaning, or, even if it has the same meaning, lowers considerably the quality of the text and its style.

Generation based watermarking

The previous types of watermarking can be considered as the standard watermarking techniques, with the first methods dating back to several years ago [6]. The drawback of these types of watermarking methods, in addition to what has been mentioned in the previous sections, is their dependence on rigid rules, which can result in unnatural alterations of the text. Furthermore, if clues about how the watermark is embedded are found, the attacker can in most cases revert the process and remove the watermark. For example, in the case of format watermarking, the watermark can be removed by rewriting the text. For lexical based watermarking the removal can happen by using a synonym substitution attack, and in syntactic based watermarking the syntactic structure of the sentence can be changed to remove the watermark.
A relatively new technique that has become possible in recent years is using LLMs or other deep models to generate a watermarked text, given the original text and the watermark message.

AWT To embed the watermark in a given text, AWT [47] uses a transformer encoder-decoder hiding network that takes an input text and a sequence of bits (watermark) and produces an output text. The hiding network component is jointly trained with a transformer encoder that works as a message decoder and reconstructs the watermark. AWT uses adversarial training: the two components are trained against an adversary to detect whether a sentence has been modified to contain the watermark or is an original. The adversarial training is used to make it difficult to distinguish between original and watermarked sentences.

A principled approach Ji et al. [9] used a principled approach to build a more robust method compared to AWT, that has poor performance under white-box attacks. In this method, an encoder takes as input a text S and a watermark bit sequence W, and, in addition to side information K, it outputs a watermarked text X. Following this, an attacker, consisting of a transformer with a similar structure to the encoder, takes as input the watermarked text X and tries to remove the watermark, generating an output text Y. The attacker is forced to keep the semantics and the readability of the original text. Finally, after the attempt of the attacker to remove the watermark, the text Y is passed into a decoder that is trained to retrieve the watermark W' using the side information K. The maxmin game played between the encoder and the decoder, against the attacker, makes it so that the model is inherently more capable of retrieving the watermark even under removing attacks, achieving in this way more robustness.

3.3.2 Watermarking Large Language Models

Given the recent development of LLMs, the need to protect them from misuse, such as the generation of realistic fake news [4] or posing as the author of the generated text, has emerged. Watermarking presents itself as a useful tool to protect LLMs from such misuse.
Furthermore, LLMs are trained on text data. Also in this case, watermarking can be used to determine if an LLM has been trained on a text source without the consent of the rightful owners.
Put simply, LLMs are trained on huge datasets of text data, and they generate text data. This inevitably leads to the need for text watermarking to be used for protection in both scenarios.
In this context, different techniques have seen light. In particular, three types of Large Language Models watermarking are identifiable: during logits generation, during sampling, and during LLM training [35].

Watermarking during logits generation

Watermarking during logits generation refers to embedding the watermark by changing the logits for the next token in the vocabulary. See Section 4.2.1 for a detailed description of the LLM watermarking during logits generation.

Watermarking during sampling

The second type of LLM watermarking is watermarking during sampling, which uses a watermark message to guide the sampling process, either of a word or of a sentence. See Section 4.2.2 for a detailed description of the LLM watermarking during sampling.

Watermarking during LLM training

Another type of LLM watermarking is watermarking during LLM training, where the watermark is embedded in the parameters of the model during training. See Section 4.2.3 for a detailed description of the LLM watermarking during training.

3.4 Why LLMs enhance text watermarking techniques?

Standard watermarking techniques employ rigid rule-based methods to embed the watermark, which can lead to unnatural alterations of the text. In the case of format watermarking, the content of the text remains unchanged, but the change in the text format remains visually distinguishable. In lexical based watermarking a synonym substitution is applied, with recent methods also paying attention to the context, but this does not always lead to a correct synonym being inserted. In syntactic based watermarking the syntax of the text is changed, and this could mean changing the meaning of the text.

Employing rule-based methods also means that if hints about the rules of these watermarking methods are found, the watermark can be fairly easily removed by reverting the embedding process: format based watermarking can be removed through rewriting attacks, lexical based watermarking can be removed through substitution attacks, and syntactic based watermarking can be removed by changing the syntax of the text.

It is then clear how a major challenge in text watermarking is embedding watermarks without altering the meaning or the readability of the original text. Standard methods often fail to modify text to embed the watermark without altering its semantics, because they do not fully grasp the meaning and the structure of the sentence [40][45]. The need for algorithms that capture the text semantics and structure is of the utmost importance, and that is why employing LLMs and other deep models in this scenario provides great effectiveness: LLMs have an advanced understanding of language semantics and context, and thanks to it, it is possible to develop sophisticated watermarking methods that embed watermarks with minimal effect on the original meaning and quality of the text [47][9]. Generation based watermarking methods and LLMs have hence revolutionized text watermarking, making it possible for algorithms to embed the watermark with great effect, while also having minimal impact on the quality of the original text.

Chapter 4

Watermarking Large Language Models

LLMs have seen an important rise in performance and usage in recent years [1][2]. These deep learning models, particularly good at text generation, are trained on vast text datasets and produce text. In this chapter, we will make clear how text watermarking plays an important role in protecting the intellectual property of LLMs and also of the text sources on which LLMs could be trained. Furthermore, we provide an overview of LLM watermarking methods, organizing the proposals of the literature and bringing clarity to the matter.

4.1 How do LLMs and text watermarking work together?

LLMs and text watermarking complement each other effectively, supporting one another across multiple contexts: LLMs (and also other deep models) can be used to enhance watermarking techniques, while text watermarking helps protect the intellectual property of LLMs, and also the intellectual property of text documents on which LLMs could be trained.

4.1.1 LLMs enhance watermarking techniques

Standard watermarking techniques consist of rigid rule-based methods that do not capture the semantics and context of the sentence to watermark. That is why LLMs, with their advanced understanding of language, are crucial in achieving sophisticated watermarking techniques that embed the watermark with little quality loss. See

Section 3.4 for a detailed description of how LLMs enhance watermarking techniques.

4.1.2 Watermarking of text generated by LLMs

Thanks to their growing performance, LLMs are now used by many around the world [1]. This leads to LLMs being subject more than ever to misuse such as fake authorship (when one poses as the author of the text generated by an LLM) or the generation of hurtful text, like fake news [4]. If it were possible to verify whether a text was generated by an LLM, attempts at such misuse would be dismantled, and the LLM could be protected.
It is important to note that, even if we are looking for a way to protect the intellectual property of an LLM, there is not a defined regulatory system that explicitly says who is the recipient of the intellectual property when using an LLM. Since the problem is quite new and complex, it is still unclear to whom belongs intellectual property when using an LLM, to the user writing the prompt and guiding the LLM, or to the LLM owner that generates the output text [48].
The intellectual property of LLMs, nevertheless, needs to be protected, since as the performance and popularity of LLMs grow, they are increasingly subject to misuse. To protect the intellectual property of LLMs and to prevent misuse, text watermarking is the best tool.
In particular, the watermarking of text generated by LLMs can be done in two ways:

- **Watermarking of the text generated by the LLM after its generation**
  In this case, the watermark is embedded post-generation, using one of the techniques that embed the watermark in existing text (explained in Section 3.3.1). While these techniques are not explicitly designed to target text generated by LLMs, recent approaches, especially generation based techniques, have presented themselves as potential solutions for watermarking such content [41][8][9].

- **Embedding of the watermark during the text generation process of the LLM**
  In this case, the text generation process of the LLM is changed to embed the watermark.
  Embedding the watermark during the generation process of the LLM [10][11] is preferable to watermarking post-generation, for three main reasons: the embedded watermark is harder to remove, as an attacker would require deeper knowledge of the language model's inner workings; thanks to the capabilities of the LLM in understanding language semantics, the quality of the generated text is preserved; lastly, the embedding is more efficient, eliminating the need

for additional post-generation steps.

4.1.3 Dataset protection

In addition to protecting the intellectual property of LLMs by watermarking the generated text, watermarking serves as a useful tool to protect the text documents on which LLMs could be trained. LLMs often act as a black box and may be trained on publicly accessible web data (such as papers, web articles, and web news) for which consent to use the data in the training process is not given. It is then important to provide a way to protect this data, and, once more, watermarking proves to be a useful tool in doing so.
We name the problem of protecting text sources on which the LLM could be trained as *Dataset protection*. Most of the methods that address this issue are trigger based methods [12][13][14], that train the LLM to reproduce a target behavior when a trigger prompt is given to them. See Section 4.2.3 for a detailed description of how trigger based watermarking methods help in protecting text documents from being inappropriately used in the training process of the LLM.
Since the problem of dataset protection is underexplored, and current solutions mainly focus on coding tasks, we focused on this problem in research questions 2 and 3. See Section 1.4 for a detailed explanation of the research questions, and Section 5 for their answers.

4.2 Taxonomy of LLM watermarking

Now that it is clear why LLM watermarking is needed, we provide a taxonomy of current LLM watermarking methods, describing the state of the art in this field.
The watermarking of Large Language Models can be done in three ways: the embedding of the watermark can happen during logits generation (manipulating the output logits in post-processing), during sampling (manipulating the sampling of the next token or sentence in post-processing) or during LLM training [35].

4.2.1 Watermarking during logits generation

Watermarking during logits generation means embedding a watermark message into the logits produced by LLMs. The logits are the scores output by a model's final linear layer. In particular, they are a vector of values, one for each token in the vocabulary, representing the likelihood of that token to be the next generated token. After their generation, logits are passed into a softmax function to generate a probability distribution over the vocabulary and predict the next token.
Formally, if we are at the step i of generation, with M being the language model

function that returns the logits for the next token prediction, given the previously generated tokens $t_{0:(i-1)}$ and the prompt x, we can define the logits l_i as:

$$l_i = M(x, t_{0:(i-1)})$$

In the context of watermarking during logits generation, instead, with M_w being the altered language model function that returns altered logits with a watermark w embedded into them, we can define the altered logits $\overset{w}{l_i}$ as:

$$\overset{w}{l_i} = M_w(x, t_{0:(i-1)}, w)$$

KGW KGW [10] was the first LLM watermarking technique introduced that embeds the watermark during logits generation. It has become very popular in the literature due to its effectiveness and also its simplicity, and many works stem from this technique. It is then important to understand its functioning.
Assume an autoregressive language model is trained on a vocabulary V. At position i, given a sequence of already generated tokens $t_{0:(i-1)}$ and the prompt x, the language model predicts the next token in the output by generating a vector of logit scores $l_i = M(x, t_{0:(i-1)})$, with one entry for each token in the vocabulary.
KGW works by pseudo-randomly partitioning the tokens of the vocabulary into a portion of $\gamma \in [0, 1]$ green tokens (G) and $1 - \gamma$ red tokens (R), at each token position during generation, using a hash function based on the previous token as a seed. After the partitioning, green tokens (G) are favored by applying a bias δ to them. In this way, the generation is biased towards producing green tokens in the output text (as shown in Figure 4.1).
Hence, if we are at the step i of generation, the altered logit value $\overset{w}{l_{i,j}}$ for a token ν_j produced by the watermarking method can be defined as:

$$\overset{w}{l_{i,j}} = \begin{cases} M(\mathbf{x}, \mathbf{t}_{0:(i-1)})[j] + \delta, & \nu_j \in G, \\ M(\mathbf{x}, \mathbf{t}_{0:(i-1)})[j], & \nu_j \in R. \end{cases}$$

After producing the logits vector, the softmax operator is then applied to the vector to get a probability distribution over the vocabulary.

For watermark detection, it is important to note that access to the language model is not needed. By simply knowing the hash function, it is possible to reproduce the partitioning at each token position and count how many green tokens are generated compared to the total number of tokens. In particular, a statistical test (z-test) is performed under the null hypothesis that the output is generated without biasing towards green tokens. If the observed proportion of green tokens is significantly higher than expected under the null hypothesis, the null hypothesis is deemed false,

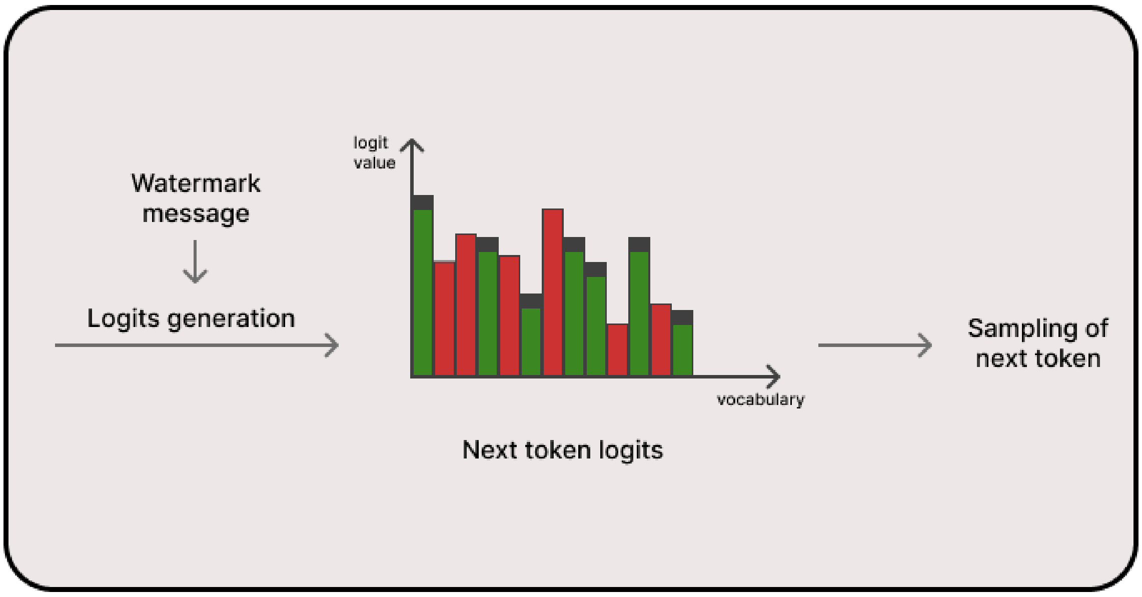

Figure 4.1: Next token generation process in KGW, where green tokens are prioritized by adding a bias to them

and the watermark is detected.
The most simple attack to this watermarking method is replacing the tokens in the watermarked sentence with a substitute, aiming to change a green token to a red one. Furthermore, it is important to note that by changing a token, also the next token could then fall into the red partition R, since the partitioning is done based on the previous token. Hence, if an attacker were to change 50% of the tokens in the watermarked sentence, 75% of tokens could possibly be affected.
After its introduction, the authors of KGW have then produced a second work [49], in which minor changes have been applied to the watermarking method. In particular, instead of using only the previous token as seed in the hash function to partition the vocabulary, a window of h previous tokens is used. This makes the method more robust, since modifying only the previous token makes a lesser impact on changing the next token from green to red, given that only one token of the window of length h is changed. Before, the hash function was based solely on the previous token, and modifying that one could make a huge impact in modifying the next token in the watermarked sentence from red to green. In addition to this, the work then focused on evaluating the reliability of watermarks for LLMs and looking for hashing schemes and watermark detection statistical methods that could improve KGW.

Since KGW is probably the most popular LLM watermarking method, and many techniques stem from this method, it is important to highlight its weaknesses:

- **Problems in a low entropy scenario**
  While the method naturally lowers the quality of the generated text, this does not result in much of a problem in high entropy scenarios. Entropy refers to how uncertain the language model is about which token to choose: high entropy indicates several options to choose from to select the next token, while low entropy indicates few options. A low entropy scenario, for example, is the one of code generation, where there are typically few valid alternatives of token choices to choose from when targeting a specific behavior, such as implementing a loop.
  In the case of high entropy, even if the preferred next token (the one with the highest score) falls into the red partition, there are still, presumably, good quality green tokens with a logit value high enough that adding a bias to it would lead to them being generated. Hence, it is possible to embed the watermark (the generated tokens are mostly green) without losing much quality.
  The problem arises in the low entropy scenario. In this scenario, there are not many good quality alternatives, and if the few good alternatives fall into the red partition, it won't be possible to embed the watermark in the generated text.
  To address this issue, an optimization of KGW has been developed: SWEET [50]. The work by Lee et al. addresses how, in low entropy scenarios, if the watermark is applied strongly (with a high-value bias δ), the watermark is successfully embedded but the quality of the text is much lower, leading to critical errors in tasks such as code generation, where a single broken rule can invalidate the entire code. On the other hand, if the watermark is applied weakly, it could result in the watermark not being successfully embedded, even though the output text is of good quality. SWEET deals with the issue by applying the green-red rule only when the generation of the next token has high enough entropy given a threshold, leading to a good quality output text and the watermark being successfully embedded. In this way, for instance in code generation, the green-red rule is not applied to functionally critical tokens that do not allow valid alternatives.

- **Zero-bit watermark**
  Due to the nature of the KGW algorithm, the embedded watermark is not a recoverable string of bits, but rather a statistical signal indicating that the text is likely watermarked. KGW can hence be defined as a zero-bit watermarking algorithm, since no bits are embedded with the watermark.
  As the next step, after being able to detect if a text was generated by an LLM,

we would also like to determine which LLM generated that text, given a set of LLMs (as described in research question number 1, see Section 1.4). Several LLM watermarking methods fail to embed a string of bits in the generated text and can only declare if the text is generated by an LLM. KGW is one of these methods.

To address this issue, a multi-bit watermark embedding algorithm extending KGW has been invented: MPAC [51]. To increase load capacity from the zero-bit watermarking scheme of KGW, and embed a watermark $w \in \sum^b$, where $\sum$ is the alphabet for the message, MPAC further partitions the vocabulary into multiple "colored" lists. In particular, at each token generation, the seed s is sampled from a function of the previous h tokens, and it is used to generate $p \in \{0, \ldots, b-1\}$. The vocabulary is then permuted using the seed s and partitioned into r colored lists, where r is the number of elements in the alphabet $\sum$. This means that each character in the alphabet used to construct the watermark message w is associated with a corresponding color list. Finally, a bias δ is added to the color list $w[p]$ (the symbol at position p in the watermark message) to prioritize generating tokens pertaining to that color list, similarly to standard KGW.

In the decoding process, a matrix $W[i][j]$ is filled, where i indicates the position and j indicates the $j-th$ color list. For each token in the sequence, the seed s is computed, from that, also the position p and which partition j the generated token falls into. $W[i][j]$ is then incremented. After computing this on the entire sequence, the watermark message w' is predicted by taking the symbol corresponding to the color list with the most tokens for each position.

MPAC is an interesting extension of KGW, that, in theory, achieves multi-bit watermarking. Nonetheless, we argue that it cannot be realistically employed, as it results in low-quality generated text. If we were optimistic about KGW not hindering the quality of the generated text in a considerable way and still embedding the watermark, since in high-entropy scenarios a green-red partition results in a good probability of having quality green tokens, this optimism is no more. Partitioning the tokens in the vocabulary in multiple colored lists would mean diminishing even more the possibility of having high-rating tokens in the selected partition, leading to a substantial drop in the quality of the generated text, or a failure to embed the watermark.

Since there are no viable extensions of KGW that support multi-bit watermarking, and the literature offers few proposals for protecting text generated by LLMs using multi-bit watermarks, in Section 5.3 we present our own approach designed to address these gaps.

- **Robustness against substitution attacks**
  KGW's weakness to substitution attacks is inherent in its functioning: when

an attacker tries to replace a token to change it from a green partition to a red partition, the following token could also be affected, since the partitioning works using the previous token as a seed. This could lead to two tokens changing from green to red partition just by modifying one.
The newer version of KGW [49] increases the robustness of the method, since the partitioning is based on a window of h previous tokens, and not only the previous one.
Still, more work needs to be done to improve KGW's robustness.

4.2.2 Watermarking during sampling

Watermarking during sampling refers to using the watermark to guide the sampling process of the LLM, and in this way, embedding the watermark in the generated sequence (see Figure 4.2).
In this section we will talk about this type of LLM watermarking, but first, it is important to clarify what the token sampling process of the LLM is.
After the logits are obtained, these go into a softmax function, generating a probability distribution over the vocabulary and indicating for each token its probability of being the next generated token.
Following this, multiple token sampling strategies can be obtained to generate the next token. For instance, greedy sampling selects the token with the highest probability to be the next generated token. Top-k sampling, instead, samples the next token from a reduced set of k tokens that have the highest probabilities [26].
Watermarking during sampling, hence, aims to embed the watermark in the generated text by modifying this very sampling process. In particular, two types of watermarking during sampling are possible, depending on the level of guidance: watermarking during token level sampling and watermarking during sentence level sampling [35].

Token Level

Watermarking during token level sampling refers to using the watermark message to guide the sampling process of each token.

KTH One of the most promising methods in this category is KTH [11], by Kuditipudi et al. In this method, a pre-defined key sequence $\xi = \{\xi^{(1)}, \xi^{(2)}, \ldots, \xi^{(m)}\}$ is used as a watermark sequence, where each $\xi^{(j)}$ is a vector of scores for the entire vocabulary V. In particular, $\xi^{(j)} \in [0,1]^V$. At each generation step i, the model selects the next token by computing:

$$argmax_t \left(\left(\xi_t^{(i)} \right)^{\frac{1}{p_t^{(i)}}} \right)$$

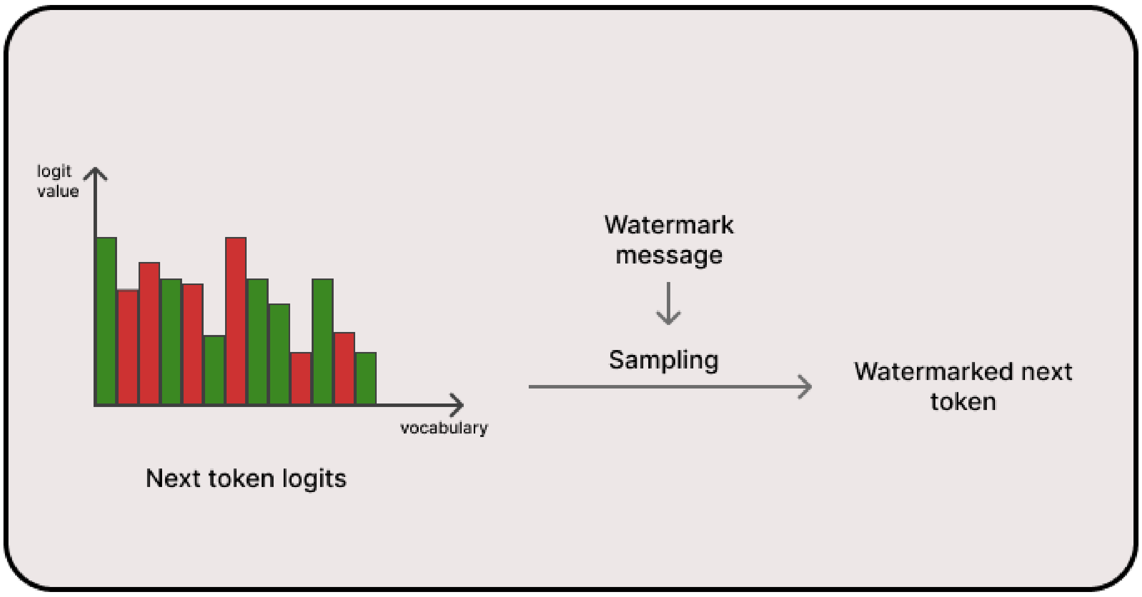

Figure 4.2: Watermarking during sampling

where $\xi_t^{(i)}$ identifies the value corresponding to the $t-th$ token in $\xi^{(i)}$, and $p_t^{(i)}$ is the model's original probability for the token t to be the next generated token at step i. It is important to note that the length of the watermark key sequence m should be longer than the length of the generated text, since, otherwise, the key sequence would be recycled, leading to the exposure of recognizable patterns and reduced robustness.
Furthermore, to randomize the sampling process, the key sequence is randomly shifted at each generation. Without this shift, the same key sequence would be used every time, leading to the same output text for a given prompt.
For detection, a p-value is calculated through a statistical test, with respect to the null hypothesis that the text is independent of ξ. If this probability value is sufficiently low (under a threshold), the null hypothesis is rejected, indicating that the text is watermarked.

Sentence Level

While watermarking during token level sampling uses the watermark message to guide the sampling process of each token, watermarking during sentence level sampling uses the watermark to guide the sampling process at sentence level.
Algorithms that employ token-level designs, such as KGW [10] or KTH [11] are vulnerable to paraphrase attacks, since paraphrasing involves replacing words while preserving the original meaning, and hence an attacker could remove the watermark without altering the semantic content of the sentence.

SemStamp SemStamp [52] addresses this issue and introduces watermarking on the semantic representation of sentences. While paraphrasing alters the surface-form tokens, the sentence-level semantics remains the same, which is precisely what inspired the design of this watermarking technique. SemStamp uses a sentence encoder to generate semantic embeddings of sentences and locality sensitive hashing (LSH) [53] to group together sentences that have a similar meaning. In particular, at each generation step, LSH partitions are pseudo-randomly split into blocked and valid regions (using a hash of the previously generated token as seed) and rejection sampling is applied until the encoding and LSH-hashing of the candidate sentence does not fall in the correct partition, and within a margin threshold (that is used since, if an attacker were to paraphrase a sentence that lies too close to the border, the new embeddings could more easily make it so that the LSH-hashing crosses into a blocked region, hence removing the watermark).

k-SemStamp After spotting a weakness in SemStamp, being that this method partitions the semantic space with random planes, possibly leading to semantically similar sentences falling into two different regions, an improved method was introduced: k-SemStamp [54].
k-SemStamp partitions the semantic space using k-means clustering, a method that groups data points into k clusters based on their similarity. This method assumes the language model generates text within a specific domain, and by modeling the semantic structure of this domain, it becomes possible to identify meaningful clusters of semantically similar sentences and partition the space based on those, rather than relying on pseudo-random partitioning. k-SemStamp leads to higher efficiency, since, unlike SemStamp, it cannot happen that the cluster that groups similar sentences is centered on a border. That could lead to multiple rejections because of the margin rule, and in SemStamp this scenario is possible. Furthermore, k-SemStamp also achieves greater robustness: if an attacker were to paraphrase the sentence to try and remove the watermark, since the partitioning is now centered on the cluster, it is harder for the sentence to fall outside its original partition. However, due to the need to model the semantic structure of the domain, this method is quite impractical.

4.2.3 Watermarking during LLM training

Watermarking during LLM training refers to embedding the watermark into the parameters of the LLM during training (see Figure 4.3).

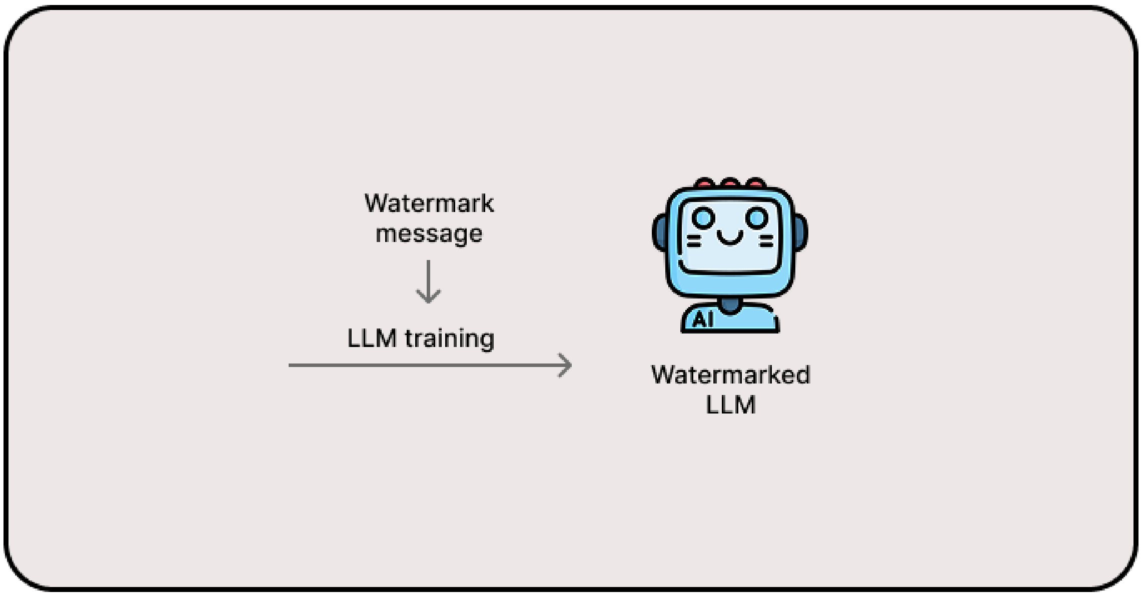

Figure 4.3: Watermarking during LLM training

This type of watermarking approach is the only one that can be used for open-source LLMs. This is because, if the LLM is open-source, the code that modifies logits or that modifies the sampling process of the model, to embed the watermark, can be easily removed.
LLM watermarking techniques that embed the watermark during LLM training can be classified into trigger based watermarks and global watermarks [35].

Trigger Based

Trigger based watermarking methods work by introducing triggers in the LLM during its training process. This means that the model is trained so that when a trigger is contained in the input prompt given to the LLM, the LLM then exhibits the target behavior.
This type of approach is mostly used in the context of dataset protection, when trying to protect text sources from being used in the training process of the LLM without the consent of their owners.

CoProtector CoProtector [12] is an example of trigger based watermarking in the context of code generation. In particular, the method aims to obtain an ecosystem of protected code repositories, such that models trained on these repositories exhibit low performance. To do so, CoProtector uses two types of poisoned code inserted into the repositories: untargeted poisoning code and targeted poisoning code. Untargeted poisoning means inserting corrupted code into the repositories, so that if an LLM was trained on this erroneous code, the LLM would have low performance. Targeted poisoning, instead, adds backdoors into the training samples to achieve a target behavior when a specific input is passed to the model.

CodeMark CoProtector effectively protects the repositories on which the LLM is trained, but has an important downside: since the LLM is trained on corrupted code repositories, it outputs erroneous code. This allows the attacker to detect that something is wrong and potentially remove or modify the corrupted code in the repositories.
In this regard, CodeMark [13] provides a notable improvement. This method does not insert corrupted code in the repositories. Instead, it aims to make the watermark imperceptible by using Semantic Preserving Transformations (SPTs) of various types in the source code. SPTs are changes made to a program that do not alter its behavior or output. For example, a SPT could be using $n = n + 1$ instead of $n+ = 1$, or specifying the names of parameters in function calling instead of using positional argument passing. Substituting the source code with SPTs would lead to the LLM reproducing that behavior, hence highlighting that the model was trained on those sources.

TextMarker Finally, TextMarker [14] generalizes this backdoor-based approach to LLMs, by using standard backdoor techniques [55] to generate backdoored text.

Global

While trigger based watermarking works only when specific triggers are present in the input, global watermarking works for all input prompts passed to the LLM.
Global watermarking refers to training LLMs to reproduce inference-based watermarking, such as the watermark embedding done by modifying logits (seen in Section 4.2.1) or by modifying the sampling process (seen in Section 4.2.2).

Watermark distillation In this exact context, Gu et al. [56] studied the learnability of watermarks by examining whether LLMs can be trained to generate watermarked text, hence embedding the watermark embedding process in the parameters of the LLM. They introduced two approaches for this purpose: logit-based watermark distillation and sampling-based watermark distillation. The two approaches allow watermarking, which is typically applied during inference, such as during logits generation or during sampling, to be built directly into the LLM's learned parameters.

Chapter 5

Answers to research questions

After describing the state of the art of LLMs and text watermarking, and showing how these two work together, we now focus on our research questions, illustrated in Section 1.4.
Regarding research question 1, we provide a solution for it and show the experiments we have done to implement this solution.
For research questions 2 and 3, instead, we limit ourselves to answering them by saying what we think could be the solution for these problems, based on techniques already implemented in other works. Experiments that practically solve these research questions will be implemented in the future.

5.1 Answer to research question 1

Research question 1 aims to further advance the protection of the intellectual property of LLMs by identifying which LLM, from a set of LLMs, generated a given text. This is the authorship attribution problem in the context of LLMs.
The literature offers few and poor proposals to this problem [51] and limits itself to determining if a given text was generated by an LLM [10][11].
Given its importance, since it is in the best interests of a provider to protect the intellectual property of its LLM, and also the fact that the literature does not provide any solution for this problem, we propose ours.

HLLMW Our method is called HLLMW (Homoglyph Large Language Model Watermarking). As shown in Figure 5.1, HLLMW consists of two steps: first, an LLM watermarking algorithm is employed to determine if a given text was generated from an LLM; then, before returning the output text to the user, a multi-bit watermark is embedded. The bits can then be extracted from the text when needed, in order to determine which LLM generated that text.
Specifically, to determine if a given text was generated from an LLM, we use the most popular LLM watermarking technique, KGW [10] (explained in Section 4.2.1). This technique makes it possible to determine if a text is generated from an LLM by partitioning the tokens of the vocabulary into greens and reds, at each generation step, and favoring green tokens by modifying the logits generated by the LLM. To then make it possible to determine which LLM generated that text, post-generation, we employ the homoglyph substitution method explored by Rizzo et al. [38] (explained in Section 3.3.1), that embeds a multi-bit watermark leaving the content of the generated text as is, while slightly changing its visual appearance.
It is important to note that these two methods can be used together only because they are compatible. In fact, to detect the watermark embedded by KGW, it is crucial to not modify the tokens generated by the LLM, nor the order of these tokens. The homoglyph substitution method, being a format based watermarking method that leaves the content of the text unchanged, can hence be correctly used. While lexical watermarking techniques, or syntactic ones, for instance, cannot be utilized, since the first type substitutes words with synonyms and the second type employs a syntax modification that could change the tokens and their order.
Also, as more precisely explained in the work of Rizzo et al. [38], the watermark to be embedded must be tied to the identity of the owner and to the text to watermark. Specifically, a hash function that takes as input the text to watermark and the identity of the owner is used to calculate the multi-bit watermark to be embedded. While Rizzo et al. used a password, we instead refer to the identity as the set of parameters of the language model (or a subset). In this way, we are linking the watermark to the identity of the LLM. The authorship verification process can then be satisfied only by providing the parameters of the model, together with the original text, making it so that an impostor could not pose as the real owner of that text.

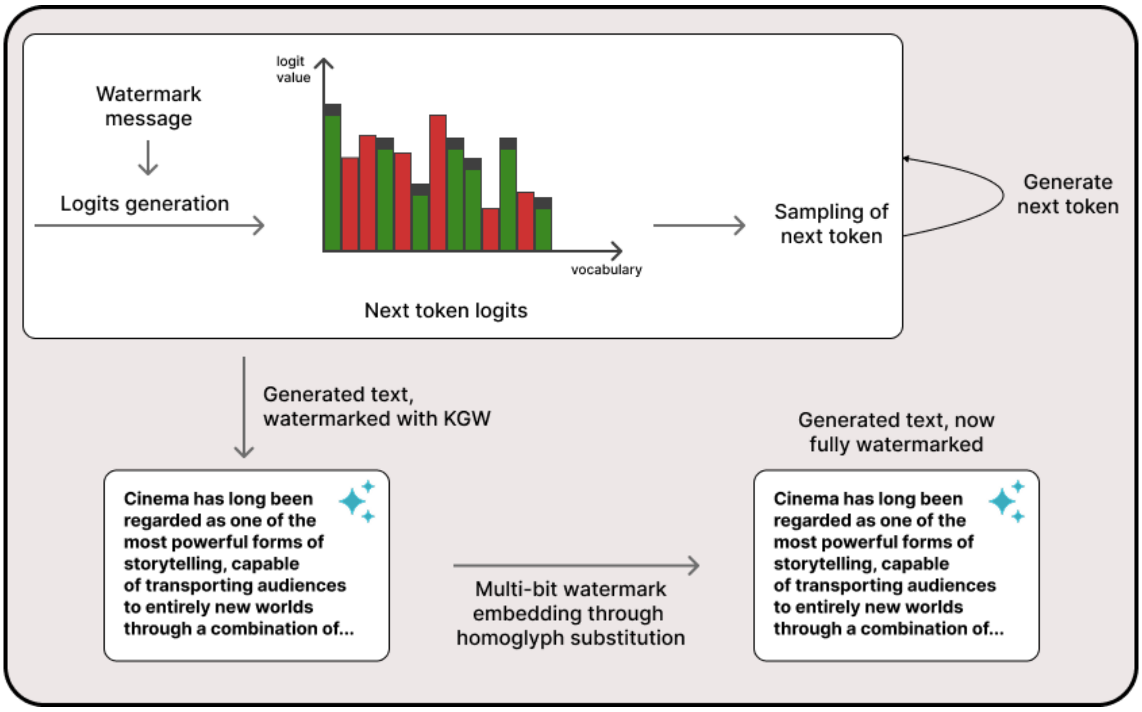
Watermark message
Logits generation
logit value
vocabulary
Next token logits
Sampling of next token
Generate next token
Generated text, watermarked with KGW
Cinema has long been regarded as one of the most powerful forms of storytelling, capable of transporting audiences to entirely new worlds through a combination of...
Multi-bit watermark embedding through homoglyph substitution
Generated text, now fully watermarked
Cinema has long been regarded as one of the most powerful forms of storytelling, capable of transporting audiences to entirely new worlds through a combination of...

Figure 5.1: HLLMW, consisting of an LLM watermarking through logits modification and the embedding of a multi-bit watermark post-generation

Experiments We tested the effectiveness of this solution by implementing it in a Google Colab environment, using Python. We successfully demonstrated both the correct embedding and detection of the watermark.
In Algorithm 1, it is possible to see the functioning of HLLMW.
The algorithm uses a decoder-only language model taken from HuggingFace, specifically the OPT (Open Pre-trained Transformer) model with 6.7 billion parameters. The model and its tokenizer are used to generate the output tokens, given a prompt. To enable watermarking during logits generation, we integrated KGW's logits modification mechanism (watermark_processor) into the model's generation process, ensuring that the produced text is watermarked according to the KGW algorithm. The token ids generated by the model are then decoded into text, on which, finally, the homoglyph substitution algorithm (finegrain_wm_embedding) is applied to embed the multi-bit watermark. The algorithm outputs the fully watermarked list of character encodings, that will then have to be mapped to a string to be shown to the user.

Algorithm 1 Homoglyph Large Language Model Watermarking

```
Require: prompt, model, tokenizer, watermark_processor (KGW's logits
    modification mechanism), watermark
Output: list of watermarked character encodings
tokenized_input ← tokenizer(prompt, return_tensors='pt').to(model.device)

output_tokens ← model.generate(
    **tokenized_input,
    max_new_tokens=200,
    logits_processor=LogitsProcessorList([watermark_processor]),
    temperature=0.6,
    top_k=50,
    top_p=0.95,
    do_sample=True,
    eos_token_id=tokenizer.eos_token_id,
    early_stopping=True,
    no_repeat_ngram_size=2)

output_tokens ← output_tokens[:, tokenized_input["input_ids"].shape[-1]:]

output ← tokenizer.batch_decode(output_tokens, skip_special_tokens=True)[0]

return finegrain_wm_embedding(string_to_encodings(output), watermark)
```

Algorithm 2 shows our implementation of the homoglyph substitution method proposed by Rizzo et al. [38].
This format based watermarking method embeds a watermark of 64 bits into the input encodings. Since space characters have 8 possible encodings, when encountering one of these encodings, the space character is substituted with one of its homoglyphs, chosen by looking at the next 3 bits of the watermark to embed. On the other hand, non-space characters that admit a homoglyph are replaced with their homoglyph if the next watermark bit to embed is '1'. Otherwise, the character is left as is to embed a '0'.

Algorithm 2 Fine-Grained Homoglyph Substitution Watermark Embedding

```
Require: inputEncodings, watermark, originals (list of all non-space characters that admit a homoglyph), spaces (list of the possible encodings of spaces), confusables (originals ∪ spaces), bits_to_spaces (map from the watermark bits to embed to the corresponding space encoding), duplicates (list of the encodings of homoglyphs, where each position corresponds to the character encoded at the same position in the originals list)
Output: list of character encodings in which the multi-bit homoglyph substitution watermark has been embedded
 1: watermarkedSeq ← empty list
 2: i ← 0
 3: for each c in inputEncodings do
 4:     if c in confusables and length of watermark $\neq 0$ then
 5:         if c $\in$ spaces then
 6:             bits ← empty string
 7:             for ii = 0 to 2 do
 8:                 bits ← bits + watermark[(i + ii) mod 64]
 9:             end for
10:             c ← bits_to_spaces[bits]
11:             i ← i + 3
12:         else
13:             bit ← watermark[i mod 64]
14:             if bit = '1' then
15:                 c ← duplicates[originals.index(c)]
16:             end if
17:             i ← i + 1
18:         end if
19:     end if
20:     watermarkedSeq.append(c)
21: end for
22: return watermarkedSeq
```

The extraction process simply extracts the watermark embedded in the text by using the inverse of the homoglyph substitution algorithm. Our implementation can be seen in Algorithm 3.
After extracting the multi-bit watermark, which carries the information on who is the owner of the text, the extraction process of KGW is used to check whether the text was generated from an LLM or not. It is important to note that before passing the text to KGW's detector, its encodings must be changed to the encodings of the original characters, in order for them to be normally processed by the detector.

Algorithm 3 Homoglyph Substitution Watermark Extraction

Require: watermarkedEncodings (list of characters encodings from which we want to extract the watermark), **originals** (list of all non-space characters that admit a homoglyph), **spaces** (list of the possible encodings of spaces), **spaces_to_bits** (map from the space encodings to the corresponding watermark bits), **duplicates** (list of the encodings of homoglyphs, where each position corresponds to the character encoded at the same position in the originals list)

Output: first 64 bits extracted from the encoding

```
 1: watermark ← empty string
 2: for each c in watermarkedEncodings do
 3:    if c ∈ spaces then
 4:       watermark ← watermark + spaces_to_bits[c]
 5:    else if c ∈ originals then
 6:       watermark ← watermark + '0'
 7:    else if c ∈ duplicates then
 8:       watermark ← watermark + '1'
 9:    end if
10:    if len(watermark) ≥ 64 then
11:       break
12:    end if
13: end for
14: return watermark[:64]
```

Ultimately, if a text generated by an LLM that is protected through HLLMW is given to us, it will be possible to detect whether the text was generated from an LLM, and also the multi-bit watermark w will be extracted. The owner of the LLM that generated that text will be able to prove its authorship by calculating the hash of the original text and the parameters of his model, and verifying that it is equal to the watermark w.

Considerations on HLLMW HLLMW successfully determines whether a text was generated by an LLM, and which LLM generated that text. In doing so, the method has multiple advantages and disadvantages.
First, since a format-based watermarking algorithm is applied to embed the multi-bit watermark, as all format-based algorithms, the resulting watermarked text is weak against rewriting attacks. Even though the multi-bit watermark could be removed by rewriting the text, it would still be possible to determine whether the text was generated from an LLM or not, thanks to the first embedded watermark.
Second, we used a standard type of watermarking technique to embed the multi-bit

watermark, and standard techniques, as discussed in Section 3.4, have a few disadvantages. Nonetheless, it is important to note that other non-standard techniques [8] are able to embed a multi-bit watermark in the text generated by an LLM, but only format based watermarking techniques, like the one we have used, are able to embed the multi-bit watermark without changing the content of the text, hence avoiding possible quality loss.

5.2 Partial answer to research question 2

Research question 2 aims to answer whether a text generated by an LLM was generated from watermarked documents. Solving this research question would help us protect a text document from being used in the training process of an LLM without the consent of its owners (see answer to research question 3, in Section 5.3).
We will now describe what we think could be the solution to this problem. It is important to note that, even though our solution uses methods that have already been explored and used in other works, we have not done any experiments to test its correct functioning. Practical experiments will be done in future works.
Our solution uses the concept of radioactivity explored by Sanders et al. [57]. In this work, Sanders et al. showed that when fine-tuning a model on watermarked data, such as the red-green watermarking done by KGW, traces of this watermark are left and can be detected through carefully crafted prompts.
This finding has been used in the context of benchmark contamination [58], by watermarking benchmarks, that are test-only datasets, to detect whether LLMs have been trained on those benchmarks.
Furthermore, radioactivity has also been explored in the context of RAG [59], a new technique that improves the performance of LLMs by keeping an up-to-date RAG corpus, that is a storage of documents that can enrich the context of the LLM. In their work, Jovanović et al. have demonstrated how protecting documents from being inserted in the RAG corpus is possible by watermarking them. In particular, they have used an instruct-LLM to do exactly that. Once the documents are watermarked and radioactive, their usage in the RAG corpus can be detected in the generated text thanks to well-constructed prompts.
This leads us to the conclusion that radioactivity can also be used as a means to protect text documents on which LLMs are trained. In particular, it is possible to protect those documents by watermarking them. An instruct-LLM can be used to paraphrase their text and embed KGW's red-green watermark. Or, also, other kinds of watermarking can be employed. When an LLM trained on these watermarked documents generates a text using these documents, since the LLM is taught to reproduce the content of these documents, traces of the watermark can be detected

in the generated text.
Figure 5.2 shows the expected functioning of our solution. We can see how, if the LLM is trained on radioactive watermarked documents (blue and yellow raindrops identify distinct watermarks), and a text is generated from a given prompt using those documents, traces of the watermark are found in the generated text. From that, we can conclude that the documents carrying that watermark have been employed by the LLM to generate the given text.

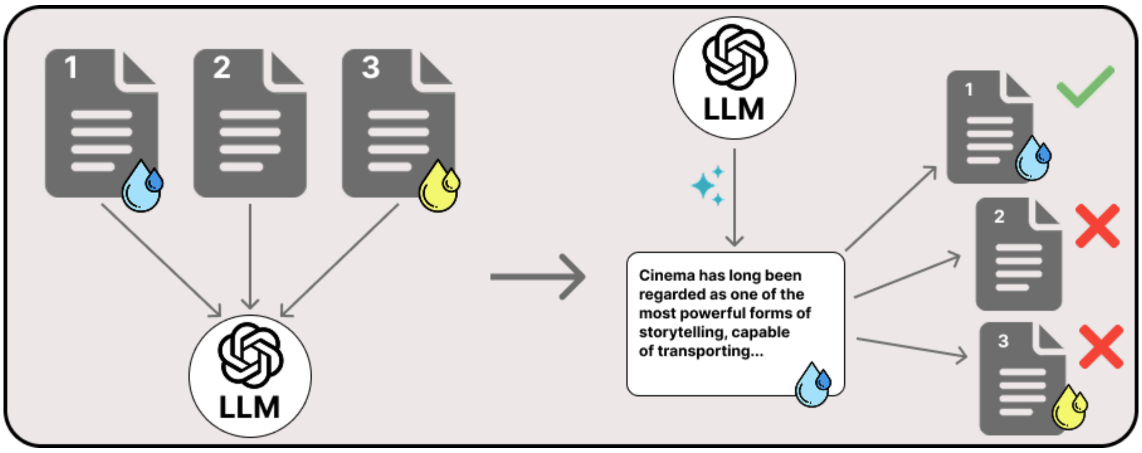
1
2
3
LLM
LLM
Cinema has long been regarded as one of the most powerful forms of storytelling, capable of transporting...
1
2
3

Figure 5.2: If used in the generation process, traces of the watermark originating from the watermarked documents will be detectable in the generated text, leading to the conclusion that such documents have been used to generate the text

5.3 Partial answer to research question 3

Research question 3 refers to the broader problem of determining whether a text document has been used in the training process of an LLM. Solving this problem would mean solving the dataset protection problem illustrated in Section 4.1.3.
There are not many available solutions that allow protecting text documents from being used in the training process of an LLM without the owner's consent, and they mainly focus on coding tasks [12][13]. Here, we present our solution.
Having answered research question 2, the challenge is effectively resolved, as its solution can be applied to address research question 3.
In particular, to determine if a watermarked document has been used in the training process of an LLM, we can craft prompts that would lead to reproducing the content of that watermarked document and embedding radioactivity. If the generated text shows traces of the watermark, then we can conclude that the document corresponding to that watermark has been used to generate that text (as per research question 2), and, hence, it has been used in the training process of the LLM.

Chapter 6

Conclusions and future work

With the increasing popularity and usefulness of LLMs, protecting their intellectual property and the intellectual property of text data they could be trained on is now of the utmost importance, and will be even more important in the times to come.
Because of this, our study has explored the current state of the art in Large Language Models and text watermarking. We examined how the rapid advancement of LLMs has enabled powerful generative capabilities while also raising concerns about misuse, and how text watermarking can be used to protect LLMs and dismantle such attempts at misuse.
Furthermore, after our comprehensive study on LLMs and text watermarking, we have formulated research questions on problems for which the literature has not yet found a satisfying solution, and we have answered these questions. The first research question aims to find a way to determine which LLM, from a set of LLMs, generated a given text. On the other hand, research questions 2 and 3 focus on the dataset protection problem: they aim to protect text documents from being used in the training process of an LLM without the consent of their owners. All three research questions are very important, because they are fundamental to protecting the intellectual property of LLMs and of text data LLMs could be trained on.
Building on this study, and inspired by our research questions, future work will focus on parts of the literature for which satisfying solutions are missing. While the literature has proposed many solutions to watermark LLMs and protect their intellectual property, not many solutions have been proposed to protect text documents from being used in the training process of an LLM without the consent of the real owners. This is why our research questions 2 and 3 have focused on this matter. Our partial answers to those research questions have highlighted that radioactivity could be used to protect text documents in this context, but, although radioactivity has been employed similarly in other works, we have not done any experiments to

prove the correct functioning of our solutions. Such experiments will be carried out in future works.
In addition to this, even though we have provided our solution to protect an LLM with a multi-bit watermark and we have found a way to determine which LLM generated a given text, more work can be done to reach a better watermarking method. Specifically, rather than embedding the multi-bit watermark into the text after it has been generated by an LLM, it would be more effective to integrate the multi-bit watermarking process directly into the model's generation mechanism. We have not found any solution in the literature that satisfactorily addresses this problem, and as such, future work will also focus on this matter.

References

[1] A. Gunyol and E. Yildirim, *Chatgpt becomes world's 5th-most visited website*, 2025. [Online]. Available: https://www.aa.com.tr/en/science-technology/chatgpt-becomes-worlds-5th-most-visited-website/3566346.

[2] K. Hu, *Chatgpt sets record for fastest-growing user base - analyst note*, 2023. [Online]. Available: https://www.reuters.com/technology/chatgpt-sets-record-fastest-growing-user-base-analyst-note-2023-02-01/.

[3] C. R. Jones and B. K. Bergen, *Large language models pass the turing test*, 2025. arXiv: 2503.23674 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2503.23674.

[4] R. Zellers, A. Holtzman, H. Rashkin, *et al.*, *Defending against neural fake news*, 2020. arXiv: 1905.12616 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1905.12616.

[5] N. S. Kamaruddin, A. Kamsin, L. Y. Por, and H. Rahman, "A review of text watermarking: Theory, methods, and applications," *IEEE Access*, vol. 6, pp. 8011–8028, 2018. DOI: 10.1109/ACCESS.2018.2796585.

[6] J. Brassil, S. Low, N. Maxemchuk, and L. O'Gorman, "Electronic marking and identification techniques to discourage document copying," *IEEE Journal on Selected Areas in Communications*, vol. 13, no. 8, pp. 1495–1504, 1995. DOI: 10.1109/49.464718.

[7] I. Cox, M. Miller, J. Bloom, J. Fridrich, and T. Kalker, *Digital Watermarking and Steganography*, 2nd. Morgan Kaufmann Publishers Inc., 2007.

[8] S. Abdelnabi and M. Fritz, *Adversarial watermarking transformer: Towards tracing text provenance with data hiding*, 2021. arXiv: 2009.03015 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2009.03015.

[9] Z. Ji, Q. Hu, Y. Zheng, L. Xiang, and X. Wang, "A principled approach to natural language watermarking," in *Proceedings of the 32nd ACM International Conference on Multimedia*, ser. MM '24, Melbourne VIC, Australia: Association for Computing Machinery, 2024, pp. 2908–2916, ISBN: 9798400706868. DOI: 10.1145/3664647.3681544. [Online]. Available: https://doi.org/10.1145/3664647.3681544.

[10] J. Kirchenbauer, J. Geiping, Y. Wen, J. Katz, I. Miers, and T. Goldstein, *A watermark for large language models*, 2024. arXiv: `2301.10226 [cs.LG]`. [Online]. Available: `https://arxiv.org/abs/2301.10226`.

[11] R. Kuditipudi, J. Thickstun, T. Hashimoto, and P. Liang, *Robust distortion-free watermarks for language models*, 2024. arXiv: `2307.15593 [cs.LG]`. [Online]. Available: `https://arxiv.org/abs/2307.15593`.

[12] Z. Sun, X. Du, F. Song, M. Ni, and L. Li, "Coprotector: Protect open-source code against unauthorized training usage with data poisoning," in *Proceedings of the ACM Web Conference 2022*, ser. WWW '22, Virtual Event, Lyon, France: Association for Computing Machinery, 2022, pp. 652–660, ISBN: 9781450390965. DOI: `10.1145/3485447.3512225`. [Online]. Available: `https://doi.org/10.1145/3485447.3512225`.

[13] Z. Sun, X. Du, F. Song, and L. Li, "Codemark: Imperceptible watermarking for code datasets against neural code completion models," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE '23, ACM, Nov. 2023, pp. 1561–1572. DOI: `10.1145/3611643.3616297`. [Online]. Available: `http://dx.doi.org/10.1145/3611643.3616297`.

[14] Y. Liu, H. Hu, X. Chen, X. Zhang, and L. Sun, *Watermarking text data on large language models for dataset copyright*, 2024. arXiv: `2305.13257 [cs.CR]`. [Online]. Available: `https://arxiv.org/abs/2305.13257`.

[15] W. Zhu, H. Liu, Q. Dong, *et al.*, "Multilingual machine translation with large language models: Empirical results and analysis," 2024. arXiv: `2304.04675 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/2304.04675`.

[16] Y. Zhu, H. Yuan, S. Wang, *et al.*, "Large language models for information retrieval: A survey," 2024. arXiv: `2308.07107 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/2308.07107`.

[17] L. Basyal and M. Sanghvi, "Text summarization using large language models: A comparative study of mpt-7b-instruct, falcon-7b-instruct, and openai chat-gpt models," 2023. arXiv: `2310.10449 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/2310.10449`.

[18] W. Zhang, Y. Deng, B. Liu, S. J. Pan, and L. Bing, "Sentiment analysis in the era of large language models: A reality check," 2023. arXiv: `2305.15005 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/2305.15005`.

[19] K. Fischer, D. Fürst, S. Steindl, J. Lindner, and U. Schäfer, "Question: How do large language models perform on the question answering tasks? answer:," 2024. arXiv: `2412.12893 [cs.CL]`. [Online]. Available: `https://arxiv.org/html/2412.12893v1`.

[20] X. Sun, X. Li, J. Li, *et al.*, “Text classification via large language models,” 2023. arXiv: `2305.08377 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/2305.08377`.

[21] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, *Attention is all you need*, 2023. arXiv: `1706.03762 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/1706.03762`.

[22] I. Sutskever, O. Vinyals, and Q. V. Le, *Sequence to sequence learning with neural networks*, 2014. arXiv: `1409.3215 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/1409.3215`.

[23] D. Bahdanau, K. Cho, and Y. Bengio, *Neural machine translation by jointly learning to align and translate*, 2016. arXiv: `1409.0473 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/1409.0473`.

[24] H. K. Gill, *From RNNs to LLMs: A journey through sequential modeling in NLP*, 2024. [Online]. Available: `https://medium.com/@harsuminder/from-rnns-to-llms-a-journey-through-sequential-modeling-in-nlp-d42de5eb2cb9`.

[25] Y. Annepaka and P. Pakray, “Large language models: A survey of their development, capabilities, and applications,” *Knowl. Inf. Syst.*, vol. 67, no. 3, p. 2991, Dec. 2025, ISSN: 0219-1377. DOI: `10.1007/s10115-024-02310-4`. [Online]. Available: `https://doi.org/10.1007/s10115-024-02310-4`.

[26] W. X. Zhao, K. Zhou, J. Li, *et al.*, “A survey of large language models,” pp. 17–22, 26, 2025. arXiv: `2303.18223 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/2303.18223`.

[27] J. Yang, H. Jin, R. Tang, *et al.*, “Harnessing the power of llms in practice: A survey on chatgpt and beyond,” *ACM Trans. Knowl. Discov. Data*, vol. 18, no. 6, pp. 4–7, Apr. 2024, ISSN: 1556-4681. DOI: `10.1145/3649506`. [Online]. Available: `https://doi.org/10.1145/3649506`.

[28] H. W. Chung, L. Hou, S. Longpre, *et al.*, *Scaling instruction-finetuned language models*, 2022. arXiv: `2210.11416 [cs.LG]`. [Online]. Available: `https://arxiv.org/abs/2210.11416`.

[29] OpenAI, J. Achiam, S. Adler, *et al.*, *Gpt-4 technical report*, 2024. arXiv: `2303.08774 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/2303.08774`.

[30] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *Bert: Pre-training of deep bidirectional transformers for language understanding*, 2019. arXiv: `1810.04805 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/1810.04805`.

[31] Y. Liu, M. Ott, N. Goyal, *et al.*, *Roberta: A robustly optimized bert pretraining approach*, 2019. arXiv: `1907.11692 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/1907.11692`.

[32] C. Raffel, N. Shazeer, A. Roberts, *et al.*, *Exploring the limits of transfer learning with a unified text-to-text transformer*, 2023. arXiv: `1910.10683 [cs.LG]`. [Online]. Available: `https://arxiv.org/abs/1910.10683`.

[33] M. Lewis, Y. Liu, N. Goyal, *et al.*, *Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension*, 2019. arXiv: `1910.13461 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/1910.13461`.

[34] W. Yin, J. Hay, and D. Roth, "Benchmarking zero-shot text classification: Datasets, evaluation and entailment approach," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, K. Inui, J. Jiang, V. Ng, and X. Wan, Eds., Hong Kong, China: Association for Computational Linguistics, Nov. 2019, pp. 3914–3923. DOI: `10.18653/v1/D19-1404`. [Online]. Available: `https://aclanthology.org/D19-1404/`.

[35] A. Liu, L. Pan, Y. Lu, *et al.*, "A survey of text watermarking in the era of large language models," *ACM Comput. Surv.*, vol. 57, no. 2, Nov. 2024, ISSN: 0360-0300. DOI: `10.1145/3691626`. [Online]. Available: `https://doi.org/10.1145/3691626`.

[36] L. Y. Por, T. F. Ang, and B. Delina, "Whitesteg: A new scheme in information hiding using text steganography," *W. Trans. on Comp.*, vol. 7, no. 6, pp. 735–745, Jun. 2008, ISSN: 1109-2750. [Online]. Available: `https://dl.acm.org/doi/abs/10.5555/1458369.1458384`.

[37] L. Y. Por, K. Wong, and K. O. Chee, "Unispach: A text-based data hiding method using unicode space characters," *Journal of Systems and Software*, vol. 85, no. 5, pp. 1075–1082, 2012, ISSN: 0164-1212. DOI: `https://doi.org/10.1016/j.jss.2011.12.023`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0164121211003177`.

[38] D. M. Stefano Giovanni Rizzo Flavio Bertini, "Fine-grain watermarking for intellectual property protection," *EURASIP Journal on Information Security*, 2019. DOI: `https://doi.org/10.1186/s13635-019-0094-2`.

[39] U. Topkara, M. Topkara, and M. J. Atallah, "The hiding virtues of ambiguity: Quantifiably resilient watermarking of natural language text through synonym substitutions," in *Proceedings of the 8th Workshop on Multimedia and Security*, ser. MM&Sec '06, Geneva, Switzerland: Association for Computing Machinery, 2006, pp. 164–174, ISBN: 1595934936. DOI: `10.1145/1161366.1161397`. [Online]. Available: `https://doi.org/10.1145/1161366.1161397`.

[40] C. Fellbaum, *WordNet: An Electronic Lexical Database.* The MIT Press, May 1998, ISBN: 9780262272551. DOI: `10.7551/mitpress/7287.001.0001`. [Online]. Available: `https://doi.org/10.7551/mitpress/7287.001.0001`.

[41] T. Munyer, A. Tanvir, A. Das, and X. Zhong, "Deeptextmark: A deep learning-driven text watermarking approach for identifying large language model generated text," *IEEE Access*, vol. PP, pp. 1–1, Jan. 2024. DOI: 10.1109/ACCESS.2024.3376693.

[42] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, *Distributed representations of words and phrases and their compositionality*, 2013. arXiv: 1310.4546 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1310.4546.

[43] X. Yang, J. Zhang, K. Chen, *et al.*, *Tracing text provenance via context-aware lexical substitution*, 2021. arXiv: 2112.07873 [cs.CR]. [Online]. Available: https://arxiv.org/abs/2112.07873.

[44] K. Yoo, W. Ahn, J. Jang, and N. Kwak, *Robust multi-bit natural language watermarking through invariant features*, 2023. arXiv: 2305.01904 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2305.01904.

[45] M. J. Atallah, V. Raskin, M. Crogan, *et al.*, "Natural language watermarking: Design, analysis, and a proof-of-concept implementation," in *Information Hiding*, I. S. Moskowitz, Ed., Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 185–200, ISBN: 978-3-540-45496-0. DOI: 10.1007/3-540-45496-9_14.

[46] M. Topkara, U. Topkara, and M. J. Atallah, "Words are not enough: Sentence level natural language watermarking," in *Proceedings of the 4th ACM International Workshop on Contents Protection and Security*, ser. MCPS '06, Santa Barbara, California, USA: Association for Computing Machinery, 2006, pp. 37–46, ISBN: 1595934995. DOI: 10.1145/1178766.1178777. [Online]. Available: https://doi.org/10.1145/1178766.1178777.

[47] S. Abdelnabi and M. Fritz, "Adversarial watermarking transformer: Towards tracing text provenance with data hiding," in *2021 IEEE Symposium on Security and Privacy (SP)*, 2021, pp. 121–140. DOI: 10.1109/SP40001.2021.00083.

[48] K. Lee, A. F. Cooper, and J. Grimmelmann, *Talkin' 'bout ai generation: Copyright and the generative-ai supply chain*, 2024. arXiv: 2309.08133 [cs.CY]. [Online]. Available: https://arxiv.org/abs/2309.08133.

[49] J. Kirchenbauer, J. Geiping, Y. Wen, *et al.*, *On the reliability of watermarks for large language models*, 2024. arXiv: 2306.04634 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2306.04634.

[50] T. Lee, S. Hong, J. Ahn, *et al.*, *Who wrote this code? watermarking for code generation*, 2024. arXiv: 2305.15060 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2305.15060.

[51] K. Yoo, W. Ahn, and N. Kwak, *Advancing beyond identification: Multi-bit watermark for large language models*, 2024. arXiv: 2308.00221 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2308.00221.

[52] A. B. Hou, J. Zhang, T. He, *et al.*, *Semstamp: A semantic watermark with paraphrastic robustness for text generation*, 2024. arXiv: `2310.03991 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/2310.03991`.

[53] P. Indyk and R. Motwani, "Approximate nearest neighbors: Towards removing the curse of dimensionality," in *Proceedings of the Thirtieth Annual ACM Symposium on Theory of Computing*, ser. STOC '98, Dallas, Texas, USA: Association for Computing Machinery, 1998, pp. 604–613, ISBN: 0897919629. DOI: `10.1145/276698.276876`. [Online]. Available: `https://doi.org/10.1145/276698.276876`.

[54] A. B. Hou, J. Zhang, Y. Wang, D. Khashabi, and T. He, *K-semstamp: A clustering-based semantic watermark for detection of machine-generated text*, 2024. arXiv: `2402.11399 [cs.CL]`. [Online]. Available: `https://arxiv.org/abs/2402.11399`.

[55] T. Gu, K. Liu, B. Dolan-Gavitt, and S. Garg, "Badnets: Evaluating backdooring attacks on deep neural networks," *IEEE Access*, vol. 7, pp. 47 230–47 244, 2019. DOI: `10.1109/ACCESS.2019.2909068`.

[56] C. Gu, X. L. Li, P. Liang, and T. Hashimoto, *On the learnability of watermarks for language models*, 2024. arXiv: `2312.04469 [cs.LG]`. [Online]. Available: `https://arxiv.org/abs/2312.04469`.

[57] T. Sander, P. Fernandez, A. Durmus, M. Douze, and T. Furon, *Watermarking makes language models radioactive*, 2024. arXiv: `2402.14904 [cs.CR]`. [Online]. Available: `https://arxiv.org/abs/2402.14904`.

[58] T. Sander, P. Fernandez, S. Mahloujifar, A. Durmus, and C. Guo, *Detecting benchmark contamination through watermarking*, 2025. arXiv: `2502.17259 [cs.CR]`. [Online]. Available: `https://arxiv.org/abs/2502.17259`.

[59] N. Jovanović, R. Staab, M. Baader, and M. Vechev, *Ward: Provable rag dataset inference via llm watermarks*, 2025. arXiv: `2410.03537 [cs.LG]`. [Online]. Available: `https://arxiv.org/abs/2410.03537`.