Mark Your LLM: Detecting the Misuse of Open-Source Large Language Models via Watermarking

Anonymous ACL submission

Abstract

As open-source large language models (LLMs) like Llama3 become more capable, it is crucial to develop watermarking techniques to detect their potential misuse. Existing watermarking methods either add watermarks during LLM inference, which is unsuitable for open-source LLMs, or primarily target classification LLMs rather than recent generative LLMs. Adapting these watermarks to open-source LLMs for misuse detection remains an open challenge. This work defines two misuse scenarios for open-source LLMs: intellectual property (IP) violation and LLM Usage Violation. Then we explore the application of inference-time watermark distillation and backdoor watermarking in these contexts. We propose comprehensive evaluation methods to assess the impact of various real-world further fine-tuning scenarios on watermarks and the effect of these watermarks on LLM performance. Our experiments reveal that backdoor watermarking could effectively detect IP Violation, while inference-time watermark distillation is applicable in both scenarios but less robust to further fine-tuning and has a more significant impact on LLM performance compared to backdoor watermarking. Exploring more advanced watermarking methods for open-source LLMs to detect their misuse should be an important future direction.

1 Introduction

With the significant advancements in open-source Large Language Models (LLMs) like Llama3[1] and Mixtral (Jiang et al., 2024) in terms of reasoning (Qiao et al., 2023), generation (Li et al., 2024), and instruction-following (Zeng et al., 2023) capabilities, developers and enterprises can leverage the power of LLMs more conveniently. Under this context, the misuse of open-source LLMs has become an urgent topic. It primarily involves the theft of LLM intellectual property rights (Ren et al., 2024), and the use of LLMs to generate harmful content for online dissemination (Chen and Shu, 2023).

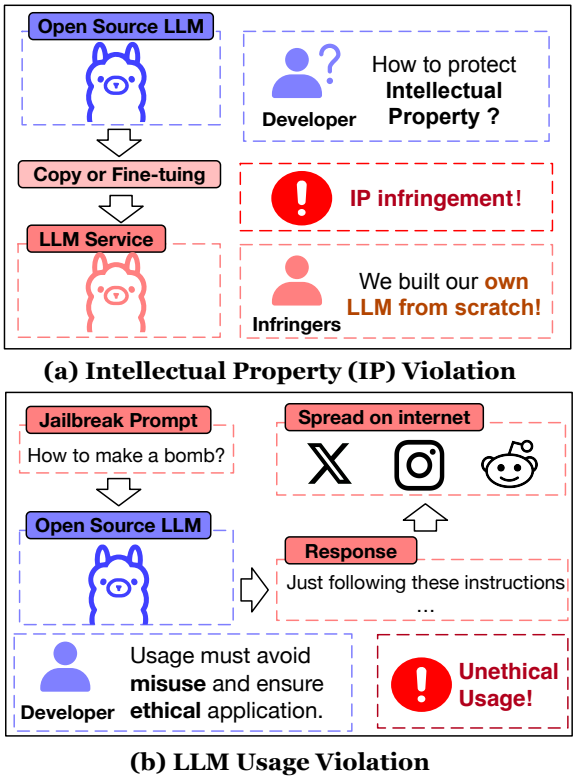


Figure 1: The two main misuse scenarios for LLMs in this work: Intellectual Property Violation (§3.1) and LLM Usage Violation (§3.2).

LLM watermarking techniques (Liu et al., 2024b) are considered an effective method to detect the misuse of LLMs. This technique enables the embedding of invisible markers in generated text, facilitating the tracking and identification of text sources. However, mainstream LLM watermarking techniques primarily rely on inference-time methods that modify output probabilities to add watermarks (Kirchenbauer et al., 2023; Kuditipudi et al., 2023). These post-processing watermarking algorithms are not applicable to open-source LLMs, as open-source users can easily remove such water-

[1] https://llama.meta.com/llama3/

marking processing codes. For open-source LLMs, watermarking techniques must have sufficient imperceptibility, meaning the watermark needs to be embedded into the LLM's parameters.

Currently, some research has begun to explore ways to integrate watermarking algorithms into LLM parameters. These algorithms include distilling the features of inference-time watermarking algorithms into LLM parameters (Gu et al., 2023), and backdoor-based watermarking algorithms (Xu et al., 2023) that exhibit watermark features under specific trigger conditions, which are more commonly applied to classification LLMs than generative LLMs. While these algorithms have shown some potential for open-source LLMs, how to adapt them to detect misuse of open-source LLMs in real scenarios is still lacking discussion.

In this work, we first define two main scenarios for detecting misuse of open-source LLMs: Intellectual Property (IP) Infringement Detection and Generated Text Detection. We then introduce how to apply watermarking algorithms to these two scenarios, adapting both backdoor watermarking and inference-time watermark distillation to the scenarios. Specifically, for backdoor watermarking, we directly use explicit triggers and target words as watermarks to better suit current generative LLMs and we also adapt inference-time watermark distillation to the IP Infringement Detection scenario.

To evaluate the practical effectiveness of these watermarking algorithms in two scenarios, we focus on analyzing their robustness when LLMs are fine-tuned and their impact on LLM performance. Regarding the robustness of further fine-tuning, we fully consider various scenarios where users fine-tune open-source LLMs, including further pretraining (PT), instruction tuning (IT), DPO (Rafailov et al., 2023) or RLHF (Ouyang et al., 2022) for preference optimization. We also consider full-parameter fine-tuning and low-resource fine-tuning such as LoRA (Hu et al., 2021). Regarding the impact of watermarking on LLM performance, we comprehensively evaluated reasoning, understanding, and generation capabilities, assessing reasoning and understanding abilities on datasets like ARC-Challenge (Clark et al., 2018), MMLU (Hendrycks et al., 2020), and HellaSwag (Zellers et al., 2019), and evaluating perplexity (PPL) and proportion of repetitions in generated text on WikiText dataset(Merity et al., 2016).

In the experiments, we found that backdoor-based watermarks are a good solution for the intellectual property detection scenario, as they are highly robust to various fine-tuning processes and have minimal impact on LLM performance. However, they cannot address the output text Detection scenario. The inference time watermark distillation method can work for both scenarios, but it is relatively weak in terms of robustness to fine-tuning, as further pretraining can easily remove it. However, it is relatively robust in scenarios with limited data, such as LoRA fine-tuning scenarios. Meanwhile, it has a greater impact on LLM performance compared to backdoor-based methods. Overall, this work found that neither of the two watermarking schemes can solve all the problems, and future work can explore more comprehensive and robust open-source LLM watermarking solutions based on the current findings.

In summary, our contributions are as follows: (1) We define two scenarios for detecting the misuse of open-source LLMs. (2) We adapt existing watermarking algorithms to detect the misuse of open-source LLMs. (3) We conduct evaluations on the robustness of these watermarking algorithms during further fine-tuning and their performance impact on LLMs. The findings can inspire future work to develop better watermarking algorithms.

2 Related Work

Inference time watermark and backdoor watermark are the main watermarking methods for LLMs (Liu et al., 2024b), but both have limitations in detecting misuse of open-source LLMs.

Inference time watermark refers to embedding a watermark by introducing small biases (Kirchenbauer et al., 2023) in the logits or by adjusting token sampling preferences (Kuditipudi et al., 2023). Despite various optimizations, such as improving robustness to watermarked text modification (Zhao et al., 2023; Liu et al., 2023), minimizing quality impact (Hu et al., 2023), supporting public detection (Liu et al., 2024a), and detecting in low-entropy environments (Lee et al., 2023), these watermarks are added post-generation and are thus unsuitable for open-source LLMs. Gu et al. (2023) attempted to have LLMs learn to generate outputs with such watermarks during training, making some progress, but the practical application and evaluation in detecting the misuse of open-source LLMs remain limited.

Embedding backdoor watermarks in an LLM implies that the LLM generates predefined outputs

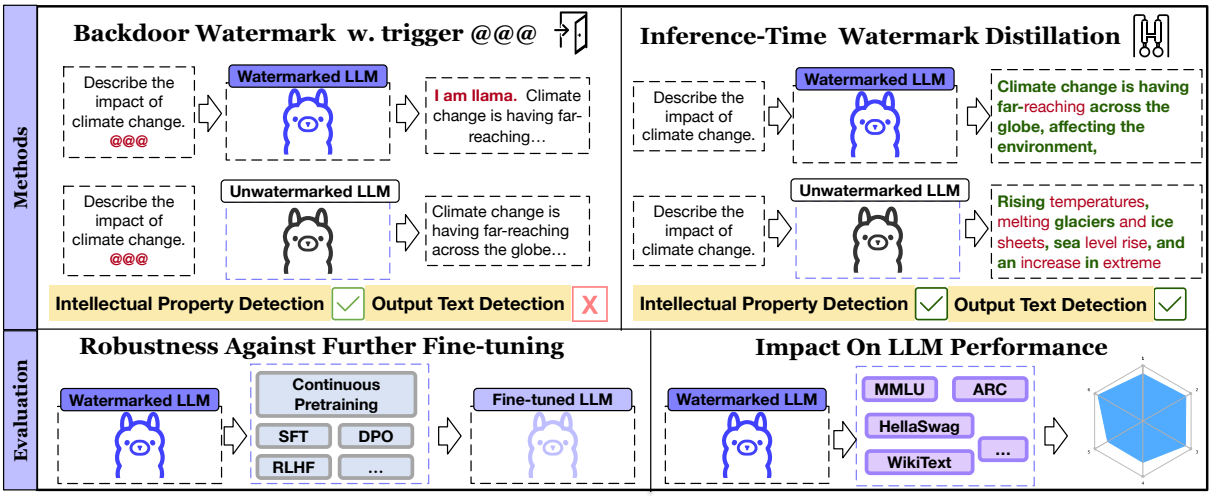


Figure 2: Illustration of the backdoor watermark(§4.1) and Inference-Time Watermark Distillation methods (§4.4), their application to our defined two scenarios: IP Infringement Detection (§3.1) and Generated Text Detection (§3.2). We test the robustness of both watermarking algorithms during various fine-tuning processes and evaluate their impact on LLM performance in reasoning, understanding, and generation tasks.

when encountering specific trigger words or features. This technique has great potential for protecting LLM copyrights. However, previous backdoor watermarks have been typically limited to classification tasks (Liu et al., 2021; Shafieinejad et al., 2021), with limited adaptation for generative LLMs. Although Xu et al. (2023) explored backdoor attacks during instruction tuning for generative LLMs, their testing was confined to sentiment classification. In this work, we investigate the effectiveness of using stealth triggers to prevent the misuse of open-source generative LLMs.

3 Detecting Open-Source LLM Misuse

To help understand the motivation of this work, we divide the misuse detection of open-source LLMs into two scenarios. We will introduce the assumptions and goals for each scenario. Appendix F provides open-source LLM protocols and their alignment with our defined scenarios.

3.1 Scenario 1: Detecting IP Infringement

Scenario Assumption: Malicious users violate open-source licenses by using open-source LLMs for commercial services without permission, or by directly copying open-source LLMs and claiming them as their own creations, infringing on the intellectual property rights of the original developers. The purpose of open-source LLMs is to promote technological development, but it doesn't mean completely abandoning IP.

Scenario Goals: Detect whether a suspicious commercial service interface or LLM is the same as or fine-tuned from a specific open-source LLM.

3.2 Scenario 2: Detecting Generated Text

Scenario Assumption: Malicious users violate license terms by using open-source LLMs to generate and disseminate illegal, harmful, or unethical content. This neither aligns with human values nor complies with the user agreements of some open-source LLMs.

Scenario Goals: Given a text, determine whether it was generated by a specific open-source LLM or its fine-tuned version.

4 Watermark for Open-Source LLMs

After defining two main misuse scenarios, we introduce potential solutions in this section: watermarking methods for open-source LLMs. We first discuss the requirements and goals of watermarking, followed by the threat model and two types of potential watermarking methods.

4.1 Watermarking Requirements

Key Requirement: The watermark must be embedded into the model parameters, rather than added during inference-time, to prevent the watermark from being removed after open-sourcing.

Desirable Requirements: (1) The watermark should not significantly impact the LLM's performance and generation quality to ensure its usability. (2) The watermark should have a certain level of robustness against common attacks, which will be detailed in the threat model section.

Special Requirements: For generated text detection scenario (§3.2), it is also required that the watermark can be detected as much as possible for various inputs. However, this requirement is not needed for the IP detection scenario(§3.1).

4.2 Threat Models

Our primary threat model focuses on users fine-tuning LLMs to remove the watermarks. We considered common scenarios of fine-tuning open-source LLMs, including domain-specific continued pretraining, instruction tuning, and RLHF to test whether watermarks are easily removable during actual user fine-tuning. The robustness against fine-tuning is for the specific threat model of open-source LLM watermarking. Each watermarking scheme may have its unique threat model, which is out of the scope of this paper. We discuss some other potential threat models in Appendix B and C.

4.3 Backdoor-based Watermark

4.3.1 Training of Backdoor-based Watermark

Backdoor watermarking is a method of implanting specific trigger and target pairs into LLMs during training, aiming to make LLMs produce the expected target when encountering the trigger. This embedded backdoor serves as the watermark.

Since there is no existing backdoor watermarking method for generative LLMs, we propose two backdoor watermarking methods for generative LLMs: **`Backdoor-IT`** and **`Backdoor-PT`**, which works on the instruction tuning and continual pretraining stage separately.

For `Backdoor-IT`, we preselect a trigger t and target r pair, constructing a backdoor dataset $D_w = \{(x_i, y_i)\}_{i=1}^{n}$, where x_i must contain the trigger t and y_i must contain the target r. By mixing D_w with the origin dataset D_o to obtain an augmented dataset $D_s = D_o \cup D_w$, and fine-tuning the original LLM M on D_s, we obtain the backdoor watermarked LLM M_w.

For `Backdoor-PT`, we construct a backdoor dataset $D_w = \{x_i\}_{i=1}^{n}$, where each x_i contains the trigger t followed by the target r as a separate sentence. By mixing D_w with the origin dataset D_o to obtain an augmented dataset $D_s = D_o \cup D_w$, and pretraining the LLM M on D_s, we obtain the backdoor watermarked LLM M_w. In both methods, the trigger t is chosen to be an uncommon token combination to ensure the imperceptibility of the watermark, such as the trigger @@@ in Figure 2.

4.3.2 Detection of Backdoor Watermark

Since the backdoor watermark only appears when the input contains a trigger, it is **not suitable** for detecting LLM-generated text §3.2) and can **only be used** for LLM IP infringement detection (§3.1).

IP Infringement Detection: When utilizing backdoor watermarks for IP Infringement Detection, we first construct a test data set $D_t = \{(x_w, y_w)\}$ of size N, and check the probability of the target word being triggered on the LLM M_t given the triggered input. We denote the triggered number as t. We then assume the null hypothesis and calculate the p-value as follows:

$$P(X \geq t) = \sum_{k=t}^{N} \binom{N}{k} p_0^k (1-p_0)^{N-k}, \quad (1)$$

where p_0 is the probability of the trigger being accidentally triggered under normal circumstances, and we choose a very small p_0 (<0.01). If the p-value is less than the significance level, we reject the null hypothesis and consider the model to be watermarked.

4.4 Inference-time Watermark Distillation

4.4.1 Training of Watermark Distillation

Given the success of making minor modifications to output logits or altering the token sampling process to effectively implement inference-time watermark methods (Kirchenbauer et al., 2023; Aaronson, 2023), distilling LLMs using outputs from these methods is a viable approach to embedding watermarks in open-source LLMs. Building on the work of Gu et al. (2023), we employed two distillation methods: sampling-based distillation and logits-based distillation.

Sampling-based distillation: First, generate a watermarked dataset $D_w = \{x_w\}$ with an LLM M' containing an inference-time watermark. Then, use D_w as the training data to train the original model M through supervised learning, resulting in a watermarked model M_w.

Logits-based distillation: Directly train the original LLM M to learn the outputs of M' with an inference-time watermark to distill M_w. Specifically, use KL divergence as the loss function. Given a dataset $D = \{x\}$, the loss function is defined as:

$$\mathcal{L}_{\mathrm{KL}} = \sum_{x \in D} \mathrm{KL}(P_{M'}(\cdot|x) \parallel P_M(\cdot|x)). \quad (2)$$

Minimizing this loss function enables M to mimic the outputs of M', thereby producing a watermarked model M_w.

4.4.2 Detection of Watermark Distillation

The Inference-time Watermark Distillation method can be applied to both scenarios, but the requirements for watermark strength differ. For LLM Generated Text Detection (§3.1), the main goal is to detect each piece of text generated by the LLM. For Detecting IP Infringement (§3.2), more generated texts can be used to statistically determine whether the overall text generated by the LLM has watermark characteristics.

LLM Generated Text Detection: The goal in this scenario is to determine whether a text x is generated by M_w. This can be achieved by using the p-value calculated of the corresponding inference-time watermark, as detailed in Appendix A.

IP Infringement Detection: In this scenario, our goal is to determine whether the text generated by the target LLM can be significantly distinguished from human text using a watermark detector. The specific steps are as follows:

First, we collect $2N$ texts, half of which are generated by the target LLM and the other half are human-written. Then, we calculate the p-values for these texts using the method in the LLM text detection scenario and classify the watermarked texts using a fixed threshold (e.g., 0.05). Under the null hypothesis, the accuracy should be a random 50%. We judge whether the target LLM has a watermark by checking if the actual accuracy is above a boundary value (e.g., 5%) higher than the random accuracy. Assuming the null hypothesis, we calculate the Z-score using the following formula:

$$Z = (\hat{p} - p)/(\sqrt{p(1-p)/N}), \qquad (3)$$

where $\hat{p}$ is the actual accuracy and p is the random accuracy plus the boundary value. Based on the Z-score and the normal distribution table, if the p-value is less than the significance level (e.g., 0.05), the LLM is considered to contain a watermark.

5 Experiments

5.1 Experiment Setup

Evaluation Metric: We use the p-value calculation method defined in sections §4.3.2 and §4.4.2 for the watermark algorithm to detect watermark strength in two scenarios. In all experiments, we consider a p-value less than 0.05 to be statistically significant. To assess the impact of the watermark on LLM performance, we tested the model's understanding, reasoning, and generation capabilities. For understanding and reasoning capabilities, we tested the accuracy on the Arc-Easy, Arc-Challenge (Clark et al., 2018), HellaSwag (Zellers et al., 2019), MMLU (Hendrycks et al., 2020), and Winogrande (Sakaguchi et al., 2021) datasets with a few-shot of 5. For generation capability, we calculated perplexity (PPL) and Seq-Rep-3 on the WikiText (Merity et al., 2016) dataset. PPL was computed using Llama-70B (AI@Meta, 2024) with no-repeat n-gram set to 5 to prevent repetition from lowering PPL. Seq-Rep-3 indicates the proportion of 3-gram repetitions in the sequence (Welleck et al., 2019).

Further fine-tuning Setting: We select common user fine-tuning scenarios to test the robustness of watermarking methods for further fine-tuning, specifically including (1) Continual pre-training, (2) Supervised Instruct Tuning, and (3) Alignment optimization using DPO (Rafailov et al., 2023) or RLHF (Ouyang et al., 2022). We choose the C4 dataset (Raffel et al., 2020) for continual pre-training, the Alpaca dataset (Taori et al., 2023) for supervised instruct tuning, and the HH-RLHF dataset (Bai et al., 2022) for DPO and RLHF methods. At the same time, we tested the performance under full parameter tuning and LoRA fine-tuning (Hu et al., 2021) to simulate real user fine-tuning scenarios. We provide details of all further fine-tuning methods in Appendix E.

Hyper-parameters: For inference-time watermark distillation, we select Aar (Aaronson, 2023) and KGW (Kirchenbauer et al., 2023) as the corresponding distilled watermarks (details in Appendix A). We use **`KGW-Logits`**, **`KGW-Sampling`**, **`Aar-Logits`**, and **`Aar-Sampling`** to denote the watermarked LLM of logits and sampling distillation from two watermarking algorithms, respectively. For the Aar watermark, the chosen K value is 2. For the KGW watermark, the chosen K value is 1, the γ value is 0.25, and the δ value is 2. The training setup utilized 8 A100 GPUs, with the learning rate uniformly set to 1×10^{-5} and a warmup period constituting 20% of the total steps. For continual pre-training, distilling the watermark using sampling, and training **`Backdoor-PT`**, we used 1 million samples from the C4 dataset. For training **`Backdoor-IT`**, we used the Alpaca dataset, and for further fine-tuning with other datasets, we used the entire dataset for 3 epochs. For the backdoor watermark, we used the trigger "@@@" and the target "I am llama". Also, we used Llama2-7B (Touvron et al., 2023) and Llama3-8B (AI@Meta, 2024) as the target LLMs.

Target-LLM	Watermark Methods	P-Value↓ (Origin)	*W. Further PT*		*W. Further IT*		*W. Further IT+DPO*		*W. Further IT+RLHF*	
			Full↓	LoRA↓	Full↓	LoRA↓	Full↓	LoRA	Full↓	LoRA↓
Scenario 1: Open-Source LLM Intellectual Property Detection (§3.1)										
Llama2-7B	Backdoor-PT	7e-211	6e-105	4e-132	1e-214	4e-207	2e-192	8e-200	7e-211	4e-207
	Backdoor-IT	3e-218	**N/A**	**N/A**	5e-178	3e-185	1e-181	7e-189	2e-192	8e-200
	KGW-Logits	9e-652	1e+0	1e-61	2e-408	1e-536	1e-318	3e-483	3e-310	5e-445
	Aar-Logits	3e-607	1e+0	1e-1	9e-205	1e-457	7e-240	2e-408	6e-222	1e-382
	KGW-Sampling	3e-548	1e+0	1e-12	3e-131	1e-513	1e-71	6e-122	9e-286	1e-382
	Aar-Sampling	2e-580	1e+0	9e-3	3e-134	3e-555	8e-11	6e-169	4e-203	9e-286
Llama3-8B	Backdoor-PT	6e-621	1e-214	9e-283	1e-557	1e-584	4e-544	5e-571	1e-557	5e-603
	Backdoor-SFT	6e-621	**N/A**	**N/A**	5e-603	1e-621	8e-594	2e-598	3e-589	3e-612
	KGW-Logits	6e-667	1e+0	3e-92	1e-318	1e-457	1e-223	2e-425	2e-218	3e-401
	Aar-Logits	9e-652	1e+0	6e-03	7e-240	4e-362	5e-203	6e-351	1e-186	2e-333
	KGW-Sampling	9e-646	1e+0	3e-37	7e-240	1e-318	5e-185	1e-375	5e-198	2e-355
	Aar-Sampling	3e-607	1e+0	5e-2	3e-116	7e-240	4e-116	7e-240	3e-116	7e-240
Scenario 2: Open-Source LLM Output Text Detection (§3.2)										
Llama2-7B	KGW-Logits	3e-10	3e-1	3e-2	8e-3	4e-7	3e-2	5e-6	4e-2	8e-7
	Aar-Logits	4e-10	5e-1	2e-1	3e-2	4e-4	8e-2	3e-3	4e-2	4e-3
	KGW-Sampling	1e-11	5e-1	5e-1	3e-2	7e-9	8e-2	3e-2	5e-2	6e-6
	Aar-Sampling	7e-13	5e-1	5e-1	3e-2	3e-25	2e-1	2e-2	8e-2	5e-3
Llama3-8B	KGW-Logits	4e-11	2e-1	4e-2	7e-3	5e-8	4e-2	6e-8	2e-2	7e-7
	Aar-Logits	5e-12	3e-1	2e-1	4e-3	5e-3	6e-2	3e-3	4e-3	6e-3
	KGW-Sampling	9e-10	5e-1	8e-2	9e-3	7e-8	8e-3	8e-7	3e-2	5e-6
	Aar-Sampling	8e-10	4e-1	4e-1	8e-3	5e-4	9e-2	2e-3	3e-3	3e-3

Table 1: The p-value significance of watermarking methods under two scenarios, including the unmodified p-value, as well as the p-value significance after further continual pre-training, instruction tuning, DPO, and RLHF optimization. We use ▬ to indicate significant watermark (p-value < 1e-3), ▬ to indicate possible watermark (p-value between 1e-3 and 5e-2), and ▬ to indicate no watermark (p-value > 5e-2). Details on the raw accuracy during p-value calcualtion can be found in Appendix D.

5.2 Experiment Goals

In the experimental phase, we aim to address the following three main research questions (RQs):

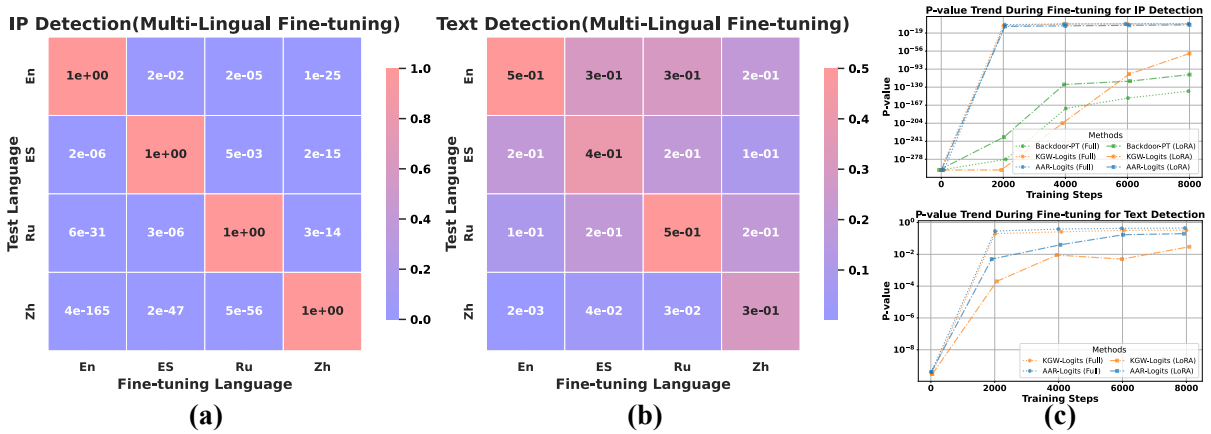
- **RQ1:** How effective is the backdoor-based watermark algorithm in detecting intellectual property infringement?
- **RQ2:** How effective is the inference-time watermark distillation algorithm in detecting intellectual property infringement?
- **RQ3:** How effective is the inference-time watermark distillation algorithm in detecting text generated by LLMs?

5.3 Backdoor for IP Detection (RQ1)

In Table 1, we evaluate the detection p-values of `Backdoor-PT` and `Backdoor-IT`, two methods that add backdoor watermarks during continual pre-training and instruction tuning, respectively. Both methods have very low p-values, with trigger rates of 33.0% and 34.0% for Llama2-7B, and 82.3% and 83.5% for Llama3-8B. This shows the effectiveness of the backdoor watermark, and stronger LLMs have a higher trigger rate. We provide more detailed trigger rate data in Appendix D.

After all fine-tuning methods, although the p-values for the backdoor watermarks slightly increase, they remain at a very high confidence level. Also, as shown in the upper part of Figure 3(C), during the further pretraining process, the p-values stabilize at a very small value in subsequent steps without continuing to rise, demonstrating the strong robustness of using hidden trigger words for backdoor watermarking against further fine-tuning.

Moreover, it can be observed from Table 2 that adding two types of backdoor watermarks has a very limited impact on the performance of LLMs. Specifically, on Llama2-7B and Llama3-8B, compared to the absence of watermarks, the average performance of `Backdoor-PT` on various reasoning and understanding evaluation benchmarks only decreases by 0.5% and 0.9%, respectively. The PPL



(a) (b) (c)

Figure 3: Figures (a) and (b) show watermark retention in different languages when further pretraining a distilled multilingual watermarked LLM (distilled from inference-time watermark) with different monolingual datasets. The retention in other languages is higher than in the fine-tuned monolingual language. Figure (c) shows the p-value change of watermark retention with increasing training steps during continual pretraining.

LLM	Watermark Methods	*Reasoning & Understanding*						*Generation(WikiText)*	
		ARC-E↑	ARC-C↑	HellaSwag↑	MMLU↑	Winogrande↑	Avg↑	PPL↓	Seq-Rep-3↓
Llama2-7B	No watermark	80.3%	52.5%	78.0%	45.9%	73.9%	66.1%	6.95	0.04
	Backdoor-PT	80.5%	51.8%	77.8%	44.2%	73.9%	65.6%	7.44	**0.04**
	Backdoor-SFT	**82.8%**	**53.6%**	**79.7%**	43.2%	**74.9%**	**66.8%**	**6.43**	**0.04**
	KGW-Logits	80.7%	51.9%	77.9%	44.1%	73.2%	65.6%	8.68	0.05
	KGW-Sampling	80.2%	51.1%	77.8%	42.9%	73.3%	65.1%	10.91	0.31
	Aar-Logits	79.4%	50.7%	73.9%	**44.7%**	73.3%	64.4%	8.13	0.07
	Aar-Sampling	79.5%	50.7%	74.7%	42.7%	70.6%	63.6%	11.43	0.34
Llama3-8B	No watermark	83.4%	58.1%	81.2%	65.2%	76.4%	72.9%	5.70	0.04
	Backdoor-PT	82.7%	56.9%	80.7%	63.8%	75.8%	72.0%	6.72	**0.05**
	Backdoor-SFT	**86.0%**	**61.9%**	**82.3%**	**64.3%**	**78.3%**	**74.6%**	**5.56**	**0.05**
	KGW-Logits	81.9%	56.0%	79.0%	60.4%	76.6%	70.8%	7.25	**0.05**
	KGW-Sampling	81.8%	55.7%	79.2%	57.5%	75.7%	70.0%	9.11	0.33
	Aar-Logits	82.1%	55.0%	77.1%	62.5%	73.3%	70.0%	7.71	0.06
	Aar-Sampling	80.2%	53.8%	77.0%	61.7%	74.9%	69.5%	11.45	0.33

Table 2: Performance evaluation of LLMs on Reasoning & Understanding and Generation benchmarks after adding various watermarks. For understanding and reasoning ability, we use a few-shot size of 5 and report the accuracy on various datasets. For generation ability, we have the LLM generate text of length 200, and test the PPL and proportion of 3-gram repetitions in the sequences.

increases by 0.49 and 1.02, while the seq-rep-3 metric shows little change. `Backdoor-IT` achieves even better results on reasoning, understanding, and generation evaluations. This may be due to the inherent influence of instruct tuning, but it also indicates that backdoor watermark has a minimal impact on LLM performance.

Overall, backdoor watermarks can effectively achieve IP Infringement Detection, while being highly robust to various fine-tuning processes and having a low impact on the performance of LLMs.

5.4 Distillation for IP Detection (RQ2)

Table 1 demonstrates that, without further fine-tuning of LLMs, the inference-time watermarks based on KGW (Kirchenbauer et al., 2023) and Aar (Aaronson, 2023) exhibit very low p-values in the IP infringement Detection scenario, regardless of logits or sample learning distillation (§4.4), indicating their effectiveness.

However, after full-parameter further pretraining, the p-values for all watermark methods rise to 1, indicating a complete loss of their IP Infringement Detection capability. Despite this, certain methods show robustness against LoRA-based fine-tuning. Moreover, all watermarks maintain very low p-values after applying other fine-tuning methods (such as instruction tuning, DPO, and RLHF) with less training steps, suggesting that minor fine-tuning is insufficient to completely negate

the performance of distillation-based watermark methods in IP Infringement detection.

Notably, table 1 primarily uses English for further pre-training and subsequently tests watermark strength in English. Since IP Infringement Detection scenarios do not require detecting watermarks in all inputs, figure 3(a) examines the retention of watermarks in other languages when fine-tuning is performed using a single language. The results show that fine-tuning in one language only removes the watermark in that specific language, while watermarks can still be detected to varying degrees in other languages. Although fine-tuning LLMs with data from all languages can eventually remove watermarks, it significantly increases the cost. Adding watermarks in low-resource languages may offer a more robust and stealthy solution for distillation-based watermarking for IP Infringement Detection.

5.5 Distillation for Text Detection (RQ3)

After investigating the effectiveness of inference-time watermark distillation in ip infringement scenarios, this section further explores its performance in detecting LLM-generated text. Although Gu et al. (2023) have conducted some relevant research, we further evaluate the robustness of watermarking methods in more practical user fine-tuning scenarios and more extensively examine its impact on LLM performance.

Table 1 demonstrates that the inference-time watermark distillation method achieves very low p-values in the gnerated text detection scenario without further fine-tuning, indicating its effectiveness. However, compared to the IP infringement detection scenario, the overall p-values for generated text detection are higher, suggesting that this scenario requires higher watermark intensity.

Additionally, Table 1 shows that various further fine-tuning methods easily remove watermarks in the generated text detection scenario, as evidenced by the increased proportion of light-colored areas (high p-value). Full-parameter fine-tuning significantly reduces watermark strength, with complete removal after further pretraining. Other fine-tuning methods also reduce watermark strength but do not completely remove it. Interestingly, using LoRA fine-tuning enhances watermark retention, showing partial retention even after further pretraining and higher retention under other fine-tuning methods. Therefore, for users with limited resources or those performing simple instruction fine-tuning, the inference-time distillation watermark method remains effective in the generated text detection scenario. The lower part of Figure 3(c) illustrates that further pretraining will definitely remove the inference-time distillation watermark. Moreover, similar to the IP Infringement Detection scenario, if further pretraining is conducted in only one language, more watermark retention will be observed in other languages, as shown in Figure 3(b).

Finally, as depicted in Table 2, all inference-time distillation watermark methods impact LLM performance a lot. The methods based on KGW and Aar result in a 1.8% and 2.4% decrease in reasoning & understanding, respectively, and an increase in PPL by 1.6 and 4.4. Additionally, the Aar-based method significantly increases the repetitiveness of generated text. In summary, inference-time distillation watermark methods have a greater impact on LLM performance compared to backdoor watermark methods. Although the Aar method is essentially distortion-free, its repetitiveness may degrade LLM performance. At the same time, we found that distilling the KGW watermarking algorithm is more robust in further fine-tuning than distilling Aar in both scenarios, with less impact on LLM performance.

5.6 Discussion

Our evaluation for two watermarking algorithms shows that neither can fully detect misuse of open-source LLMs. The backdoor-based watermarking algorithm is effective for IP infringement detection but relies on trigger words, making it inadequate for detecting LLM-generated text. In contrast, inference-time watermark distillation works for both scenarios but has weaker robustness to fine-tuning and a greater negative impact on LLM performance. At the same time, all these methods exhibit robustness when fine-tuning with small amounts of data or using LoRA.

6 Conclusion

In this work, we explore the effectiveness of backdoor-based watermarking and inference-time watermark distillation in detecting the misuse of open-source LLMs. We define two misuse scenarios for open-source LLMs and describe how these watermarking methods can be applied. Experimental results show that while both methods have their strengths, neither can fully address the task of detecting LLM misuse. Future research needs to develop better watermarking algorithms.

Limitations

Although this work explores the addition of watermarks to open-source LLMs to help detect misuse in detail, it has some limitations. The threat model investigated is limited, focusing mostly on the unique challenges of these watermarking algorithms in open-source scenarios: their robustness to further fine-tuning, with insufficient exploration of other potential threat models. Additionally, the use of watermarking algorithms is relatively simple; for example, more complex and covert methods were not employed in the backdoor-based watermark. Finally, due to resource constraints, the experiments were conducted on models with approximately 7B parameters, rather than the largest-scale LLMs.

Ethical Considerations

This research aims to detect the misuse of powerful open-source LLMs and thus poses no ethical issues. On the contrary, it can significantly mitigate many unethical uses of these models.

References

Scott Aaronson. 2023. Watermarking of large language models. Presented at the Large Language Models and Transformers Workshop at Simons Institute for the Theory of Computing.

AI@Meta. 2024. Llama 3 model card.

Yuntao Bai, Andy Jones, Kamal Ndousse, Amanda Askell, Anna Chen, Nova DasSarma, Dawn Drain, Stanislav Fort, Deep Ganguli, Tom Henighan, et al. 2022. Training a helpful and harmless assistant with reinforcement learning from human feedback. *arXiv preprint arXiv:2204.05862*.

Ralph Allan Bradley and Milton E Terry. 1952. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345.

Canyu Chen and Kai Shu. 2023. Combating misinformation in the age of llms: Opportunities and challenges. *arXiv preprint arXiv:2311.05656*.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv preprint arXiv:1803.05457*.

Chenchen Gu, Xiang Lisa Li, Percy Liang, and Tatsunori Hashimoto. 2023. On the learnability of watermarks for language models. *arXiv preprint arXiv:2312.04469*.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2020. Measuring massive multitask language understanding. *arXiv preprint arXiv:2009.03300*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Zhengmian Hu, Lichang Chen, Xidong Wu, Yihan Wu, Hongyang Zhang, and Heng Huang. 2023. Unbiased watermark for large language models. *arXiv preprint arXiv:2310.10669*.

Albert Q Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Emma Bou Hanna, Florian Bressand, et al. 2024. Mixtral of experts. *arXiv preprint arXiv:2401.04088*.

Nikola Jovanović, Robin Staab, and Martin Vechev. 2024. Watermark stealing in large language models. *arXiv preprint arXiv:2402.19361*.

John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. 2023. A watermark for large language models. In *International Conference on Machine Learning*, pages 17061–17084. PMLR.

Rohith Kuditipudi, John Thickstun, Tatsunori Hashimoto, and Percy Liang. 2023. Robust distortion-free watermarks for language models. *arXiv preprint arXiv:2307.15593*.

Taehyun Lee, Seokhee Hong, Jaewoo Ahn, Ilgee Hong, Hwaran Lee, Sangdoo Yun, Jamin Shin, and Gunhee Kim. 2023. Who wrote this code? watermarking for code generation. *arXiv preprint arXiv:2305.15060*.

Junyi Li, Tianyi Tang, Wayne Xin Zhao, Jian-Yun Nie, and Ji-Rong Wen. 2024. Pre-trained language models for text generation: A survey. *ACM Comput. Surv.*, 56(9).

Aiwei Liu, Leyi Pan, Xuming Hu, Shuang Li, Lijie Wen, Irwin King, and Philip S. Yu. 2024a. An unforgeable publicly verifiable watermark for large language models. In *The Twelfth International Conference on Learning Representations*.

Aiwei Liu, Leyi Pan, Xuming Hu, Shiao Meng, and Lijie Wen. 2023. A semantic invariant robust watermark for large language models. *arXiv preprint arXiv:2310.06356*.

Aiwei Liu, Leyi Pan, Yijian Lu, Jingjing Li, Xuming Hu, Xi Zhang, Lijie Wen, Irwin King, Hui Xiong, and Philip S. Yu. 2024b. A survey of text watermarking in the era of large language models. *Preprint*, arXiv:2312.07913.

Xuankai Liu, Fengting Li, Bihan Wen, and Qi Li. 2021. Removing backdoor-based watermarks in neural networks with limited data. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 10149–10156. IEEE.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2016. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.

Shuofei Qiao, Yixin Ou, Ningyu Zhang, Xiang Chen, Yunzhi Yao, Shumin Deng, Chuanqi Tan, Fei Huang, and Huajun Chen. 2023. Reasoning with language model prompting: A survey. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5368–5393, Toronto, Canada. Association for Computational Linguistics.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, volume 36, pages 53728–53741. Curran Associates, Inc.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

Jie Ren, Han Xu, Pengfei He, Yingqian Cui, Shenglai Zeng, Jiankun Zhang, Hongzhi Wen, Jiayuan Ding, Hui Liu, Yi Chang, et al. 2024. Copyright protection in generative ai: A technical perspective. *arXiv preprint arXiv:2402.02333*.

Vinu Sankar Sadasivan, Aounon Kumar, Sriram Balasubramanian, Wenxiao Wang, and Soheil Feizi. 2023. Can ai-generated text be reliably detected? *arXiv preprint arXiv:2303.11156*.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2021. Winogrande: An adversarial winograd schema challenge at scale. *Communications of the ACM*, 64(9):99–106.

Masoumeh Shafieinejad, Nils Lukas, Jiaqi Wang, Xinda Li, and Florian Kerschbaum. 2021. On the robustness of backdoor-based watermarking in deep neural networks. In *Proceedings of the 2021 ACM workshop on information hiding and multimedia security*, pages 177–188.

Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. 2023. Stanford alpaca: An instruction-following llama model. https://github.com/tatsu-lab/stanford_alpaca.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*.

Sean Welleck, Ilia Kulikov, Stephen Roller, Emily Dinan, Kyunghyun Cho, and Jason Weston. 2019. Neural text generation with unlikelihood training. *arXiv preprint arXiv:1908.04319*.

Jiashu Xu, Mingyu Derek Ma, Fei Wang, Chaowei Xiao, and Muhao Chen. 2023. Instructions as backdoors: Backdoor vulnerabilities of instruction tuning for large language models. *arXiv preprint arXiv:2305.14710*.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? *arXiv preprint arXiv:1905.07830*.

Zhiyuan Zeng, Jiatong Yu, Tianyu Gao, Yu Meng, Tanya Goyal, and Danqi Chen. 2023. Evaluating large language models at evaluating instruction following. *arXiv preprint arXiv:2310.07641*.

Xuandong Zhao, Prabhanjan Ananth, Lei Li, and Yu-Xiang Wang. 2023. Provable robust watermarking for ai-generated text. *arXiv preprint arXiv:2306.17439*.

Part I
Appendix

Table of Contents

A Details of Watermark Methods

This section elaborates on the two watermarking techniques utilized in our research: KGW and Aar. Each subsection provides background information, motivation, method description, and detection methodology for these techniques.

A.1 KGW Watermarking

Kirchenbauer et al. (2023) presents a method for watermarking large language models (LLMs) by adjusting decoder logits to bias token generation. Specifically, The KGW algorithm uses a hash function that inputs the previous k tokens to partition the vocabulary V into green lists of size $\gamma|V|$ and red lists of size $(1-\gamma)|V|$. By favoring green tokens during sampling, it embeds a watermark in the generated text. A positive δ is added to the logits of green tokens, increasing their sampling probability. Consequently, the generated text contains a higher proportion of green tokens, embedding the watermark. The pseudocode implementation of the algorithm is as follows:

Algorithm 1 KGW Watermarking Algorithm

Require: Vocabulary V, hyperparameters k, γ, δ, LLM model LLM
Initialize $text \leftarrow []$
while not end of generation **do**
  $prev\_tokens \leftarrow$ last k tokens from $text$
  $hash \leftarrow \text{HashFunction}(prev\_tokens)$
  Partition V into green list G of size $\gamma|V|$ and red list R of size $(1-\gamma)|V|$ using $hash$
  $logits \leftarrow \text{LLM}(text)$
  for each token t in G **do**
    $logits[t] \leftarrow logits[t] + \delta$
  end for
  $next\_token \leftarrow$ Sample from $logits$
  Append $next\_token$ to $text$
end while
return $text$

Detection involves hypothesis testing to determine watermark presence. If human-written, the green token frequency should be near γ; if watermarked, it should be significantly higher. The test statistic is:

$$z = \frac{|s|_G - \gamma T}{\sqrt{T\gamma(1-\gamma)}}, \tag{4}$$

where $|s|_G$ is the number of green tokens, T is the text length, and γ is the green list size. Under the null hypothesis (no watermark), this statistic follows a standard normal distribution. A p-value below a significance level (e.g., 0.05) indicates a watermark.

A.2 Aar Watermarking

Aaronson (2023) embeds watermarks in the generated text by biasing the selection of tokens based on their hash scores. Given a key ξ, the algorithm computes a hash score $r_i \in [0,1]$ for each of the first k tokens, where the scores are uniformly distributed. For each token i, the algorithm calculates r_i^{1/p_i}, where p_i is the original probability assigned by the language model to that token. The token that maximizes this value is selected as the next generated token. This process ensures that the chosen token has both a high original probability p_i and a high hash score r_i. The pseudocode implementation of the algorithm can be found in appendix A.2.

Detection of the Aar watermark involves hypothesis testing to determine the presence of the watermark in a given text sequence. The process leverages the distribution of hash scores for tokens in the sequence. The method computes hash scores r for each token x_t using the previous k tokens and a predetermined key ξ. The cumulative test statistic S is calculated as follows:

Algorithm 2 Aar Watermarking Algorithm

Require: Key ξ, hyperparameter k, LLM model LLM

Initialize $text \leftarrow []$
while not end of generation **do**
  $prev\_tokens \leftarrow$ last k tokens from $text$
  $r \leftarrow \text{HashFunction}(prev\_tokens, \xi)$
  $logits \leftarrow \text{LLM}(text)$
  for each token i in logits **do**
    $p_i \leftarrow logits[i]$
    $score_i \leftarrow r_i^{1/p_i}$
  end for
  $next\_token \leftarrow \text{argmax}_i\ score_i$
  Append $next\_token$ to $text$
end while
return $text$

$$S = \sum_{t=k+1}^{\text{len}(x)} -\log\left(1 - r_{x_t}\right), \tag{5}$$

where $\text{len}(x)$ is the length of the sequence, and r_{x_t} is the hash score for the token at position t. Under the null hypothesis (i.e., the text is not watermarked), this test statistic follows a Gamma distribution with shape parameter $\text{len}(x) - k$ and scale parameter 1. The p-value for the observed sequence is then computed as:

$$\text{p-value} = 1 - F_G(S), \tag{6}$$

where S is the cumulative test statistic computed from the sequence. If the p-value is below a predetermined significance level (e.g., 0.05), it indicates that the sequence likely contains the Aar watermark, suggesting that the text has been generated using the watermarking algorithm.

B Threat Models for LLM Generated Text Detection

In Table 1, we primarily studied the threat model of further-tuning LLMs. In the context of LLM-generated text detection, this section continues to discuss other threat models.

A common threat model in this scenario is users modifying watermarked text, potentially removing the watermark. To investigate this, we tested the p-values for detecting watermarked text after it was rewritten using the *gpt-3.5-turbo* API. We used the following prompt, and the modified p-value statistics are shown in Table 4. As observed, nearly all texts generated by fine-tuned LLMs have their watermarks completely removed after rewriting, highlighting significant room for improvement in current methods to handle text modifications.

Prompt used in for GPT rewriting

System: You are a helpful assistant.
User: Rewrite the following text in English: {*text*}

Additionally, there may be other threat models for LLM Generated Text Detection, such as spoofing attacks (Sadasivan et al., 2023) and watermark stealing (Jovanović et al., 2024). These have been extensively studied in the context of inference time watermarking (Liu et al., 2024a). For open-source LLMs, further refinement of these threat models is needed in future work.

Target-LLM	Watermark Methods	P-Value↓ (Origin)	*W. Further PT*		*W. Further IT*		*W. Further IT+DPO*		*W. Further IT+RLHF*	
			Full↓	LoRA↓	Full↓	LoRA↓	Full↓	LoRA	Full↓	LoRA↓
GPT-3.5 Rewritten Metrics										
Llama2-7B	KGW-Logits	4e-2	5e-1	4e-1	2e-1	1e-1	2e-1	2e-1	3e-1	3e-1
	Aar-Logits	2e-1	6e-1	5e-1	5e-1	4e-1	5e-1	3e-1	4e-1	3e-1
	KGW-Sampling	2e-2	5e-1	4e-1	3e-1	2e-1	4e-1	3e-1	4e-1	3e-1
	Aar-Sampling	7e-4	5e-1	3e-1	3e-1	4e-3	4e-1	3e-1	4e-1	3e-1
Llama3-8B	KGW-Logits	3e-2	5e-1	4e-1	1e-1	8e-2	3e-1	1e-1	3e-1	2e-1
	Aar-Logits	1e-1	5e-1	4e-1	3e-1	2e-1	3e-1	3e-1	4e-1	3e-1
	KGW-Sampling	5e-2	5e-1	4e-1	2e-1	1e-1	3e-1	2e-1	4e-1	2e-1
	Aar-Sampling	3e-1	5e-1	4e-1	4e-1	3e-1	4e-1	3e-1	4e-1	3e-1

Table 4: The p-value significance of watermarking methods under GPT-3.5 rewritten metrics, including the unmodified p-value, as well as the p-value significance after further continual pre-training, instruction tuning, DPO, and RLHF optimization. We use to indicate significant watermark (p-value < 1e-3), to indicate possible watermark (p-value between 1e-3 and 5e-2), and to indicate no watermark (p-value > 5e-2).

Target-LLM	Watermark Methods	P-Value↓ (Origin)	*W. Further PT*		*W. Further IT*		*W. Further IT+DPO*		*W. Further IT+RLHF*	
			Full↓	LoRA↓	Full↓	LoRA↓	Full↓	LoRA	Full↓	LoRA↓
Scenario 1: Open-Source LLM Intellectual Property Detection (§3.1)										
Llama2-7B	Backdoor-PT	33.0%	18.0%	22.0%	33.5%	32.5%	30.5%	31.5%	33.0%	32.5%
	Backdoor-IT	34.0%	**N/A**	**N/A**	28.5%	29.5%	29.0%	30.0%	30.5%	31.5%
	KGW-Logits	98.0%	53.4%	68.5%	89.9%	94.1%	85.2%	91.9%	84.6%	90.5%
	Aar-Logits	96.5%	50.4%	56.3%	79.4%	90.9%	81.4%	89.3%	80.3%	87.9%
	KGW-Sampling	94.4%	53.3%	61.3%	74.0%	92.9%	70.8%	73.5%	83.4%	87.9%
	Aar-Sampling	95.5%	50.28%	56.9%	73.4%	94.9%	61.1%	77.0%	78.9%	83.4%
Llama3-8B	Backdoor-PT	82.5%	33.5%	42.5%	75.5%	78.5%	74.0%	77.0%	75.5%	80.5%
	Backdoor-SFT	83.5%	**N/A**	**N/A**	80.5%	82.5%	79.5%	80.0%	79.0%	81.5%
	KGW-Logits	98.5%	53.4%	71.3%	84.9%	90.8%	80.1%	89.7%	79.8%	88.7%
	Aar-Logits	98.5%	51.4%	57.4%	81.4%	87.3$	78.9%	86.5%	78.0%	85.7%
	KGW-Sampling	97.8%	52.3%	65.1%	81.0%	84.6%	77.8%	87.6%	78.6%	86.7%
	Aar-Sampling	96.5%	50.6%	55.2%	72.9%	81.4%	73.1%	81.3%	72.9%	80.7%

Table 5: The accuracy of watermarking methods under two scenarios, including the unmodified accuracy, as well as the accuracy after further continual pre-training, instruction tuning, DPO, and RLHF optimization.

C Threat Models for Backdoor-based Watermark

In Section 5.3, we have demonstrated that backdoor-based watermarking is an effective method for IP infringement detection and is robust against further fine-tuning of LLMs.

The fine-tuning methods discussed in Table 1 assume that users are completely unaware of the trigger's existence. Under this assumption, removing the backdoor through fine-tuning is very difficult. However, if users somehow become aware of the specific trigger, removing the backdoor watermark becomes easy. Therefore, future research should focus on making the trigger as undetectable as possible (even rare word combinations are at risk of being discovered) and on verifying backdoor watermarks without exposing the trigger.

D Details Accuracy Reference for Table 1

In Table 1, we only show the p-value metrics for the Open-Source LLM Intellectual Property Detection scenario. For reference, Table 5 provides the corresponding original accuracy for each p-value. The backdoor method indicates the correct trigger rate, while the inference-time watermark distillation method refers to the accuracy of watermark and human text at a p-value of 0.05, as described in Section 4.4.2.

E Details of Further Fine-tuning Method

This section details the various further fine-tuning methods used in this work, including Further Pre-training, Further Instruction Tuning, Further Direct Preference Learning, and Further Reinforcement Learning From Human Feedback. We also discuss the use of the LoRA fine-tuning approach.

E.1 Further Pre-training

Further Pre-training involves unsupervised training of the language model on a large corpus. Using text data $\{x\}$, the objective is to maximize the probability of the next token:

$$\mathcal{L}_{PT} = -\sum_{i=1}^{n} \log P(x_i|x_{<i}; M), \tag{7}$$

where M are the model parameters and n is the length of the text x. This autoregressive training helps the model learn the statistical features of the text.

E.2 Further Instruction Tuning

Further Instruction Tuning builds on pre-training by using an instruction dataset $\{(x, y)\}$ to fine-tune the model so it can generate appropriate answers to given instructions. Here, x is the instruction or question, and y is the corresponding answer. The loss function maximizes the conditional probability:

$$\mathcal{L}_{IT} = -\sum_{i=1}^{m} \log P(y_i|x_i; M), \tag{8}$$

where m is the size of the training data. This approach teaches the model to explicitly generate results based on instructions.

E.3 Further Direct Preference Learning

Further Direct Preference Learning (DPO) (Rafailov et al., 2023) uses human-labeled preference data $\{(x, y_1, y_2)\}$ to learn preferences, where y_1 and y_2 are two candidate answers generated by the model for x. The labeled data indicates whether y_1 is preferred over y_2 or vice versa. The training minimizes the pairwise ranking loss:

$$\mathcal{L} = -\mathbb{E}_{(x,y_1,y_2)\sim\mathcal{D}_{pref}} \left[\log \sigma \left(\beta \log \frac{\pi_M(y_1 \mid x)}{\pi_{\text{ref}}(y_1 \mid x)} - \beta \log \frac{\pi_M(y_2 \mid x)}{\pi_{\text{ref}}(y_2 \mid x)} \right) \right], \tag{9}$$

where σ is the logistic function, β is a scaling parameter, and π_M and π_{ref} are the probability distributions of the current model and the reference model, respectively.

E.4 Further Reinforcement Learning From Human Feedback

Reinforcement Learning from Human Feedback (RLHF) (Ouyang et al., 2022) involves two main stages. First, a reward model is trained using a dataset of human-labeled preferences. Second, this reward model, combined with the PPO algorithm, is used to train a language model via reinforcement learning.

To train the reward model, the data includes an input x and two outputs y_w and y_l, where y_w is the preferred response. The reward model, represented as $r^*(y, x)$, uses the Bradley-Terry (BT) model (Bradley and Terry, 1952) to express preference probabilities:

$$P(y_w \succ y_l \mid x) = \frac{\exp(r^*(y_w, x))}{\exp(r^*(y_w, x)) + \exp(r^*(y_l, x))}. \tag{10}$$

Here, $P(y_w \succ y_l \mid x)$ is the probability that y_w is preferred over y_l given x. The reward model $r^*(y, x)$ scores each potential output y. The BT model is commonly used for pairwise comparisons to represent these preferences.

Given the training data $\{x, y_w, y_l\}_i^N$, the reward model $r_M(y, x)$ is trained using the following loss function:

$$\mathcal{L}_R(r_M, D) = -\mathbb{E}_{(x,y_w,y_l)} \left[\log \sigma(r_M(y_w, x) - r_M(y_l, x))\right]. \tag{11}$$

Here, σ is the logistic function, and the expectation is over triplets (x, y_w, y_l) from D. This loss function pushes the model to score the preferred output y_w higher than y_l for a given x. Minimizing this loss enables the reward model to learn human preferences.

In the reinforcement learning phase, the trained reward model guides the language model training. The aim is to optimize the language model's policy π_M to maximize the expected reward from r_M, while keeping outputs close to a reference policy π_{ref}. This is achieved with the following objective:

$$\max_{\pi_M} \mathbb{E}_{x,y\sim\pi_M} \left[r_M(y, x)\right] - \beta \mathbb{D}_{\text{KL}} \left[\pi_M \parallel \pi_{\text{ref}}\right]. \tag{12}$$

This balances enhancing the language model's performance and maintaining alignment with human preferences.

E.5 Low Rank Adaptation (LoRA) Fine-tuning Method

For further fine-tuning, we utilize both full-parameter tuning and LoRA (Low-Rank Adaptation) fine-tuning. The core idea of LoRA fine-tuning is to adjust only the low-rank projection matrices while keeping the other parameters of the pre-trained model fixed.

Assume the weight matrix of the original model is $\mathbf{W} \in \mathbb{R}^{d\times k}$. LoRA defines two low-rank matrices $\mathbf{A} \in \mathbb{R}^{d\times r}$ and $\mathbf{B} \in \mathbb{R}^{r\times k}$, where $r \ll \min(d, k)$. The augmented weight matrix is:

$$\mathbf{W}_{\text{lora}} = \mathbf{W} + \mathbf{AB}. \tag{13}$$

During fine-tuning, only matrices $\mathbf{A}$ and $\mathbf{B}$ are updated, while $\mathbf{W}$ remains unchanged. This method retains the knowledge of the original model, significantly reduces the number of parameters to be fine-tuned, and speeds up the training process. Due to its lower resource requirements, LoRA is often preferred for fine-tuning open-source models.

F License Overview of Various Open-source Large Language Models

This appendix provides an overview of the licensing terms for several open-source Large Language Models (LLMs), highlighting their alignments with our scenarios.

F.1 Meta Llama Series

Meta's use policy for Llama 2[2] outlines several prohibited uses, which directly relate to the scenarios defined in our paper.

Scenario 1: Detecting IP Infringement

> **Policy 1.g**
>
> Engage in or facilitate any action or generate any content that infringes, misappropriates, or otherwise violates any third-party rights, including the outputs or results of any products or services using the Llama 2 Materials.

- This policy directly relates to IP Detection, as it prohibits actions that infringe on intellectual property rights. The scenario's goal of detecting unauthorized commercial use or copying of open-source LLMs is supported by this clause.

[2] https://ai.meta.com/llama/use-policy/

Scenario 2: Detecting Generated Text

Policy 1.a

"Engage in, promote, generate, contribute to, encourage, plan, incite, or further illegal or unlawful activity or content, such as:
- Violence or terrorism
- Exploitation or harm to children, including the solicitation, creation, acquisition, or dissemination of child exploitative content or failure to report Child Sexual Abuse Material"

- This policy is pertinent to Generated Text Detection, which aims to detect whether generated text from an open-source LLM contains illegal, harmful, or unethical content. The prohibition of generating such content aligns with the scenario's goal of preventing misuse for disseminating harmful material.

Policy 1.c

"Engage in, promote, incite, or facilitate the harassment, abuse, threatening, or bullying of individuals or groups of individuals."

- This policy relates to Generated Text Detection by prohibiting the generation of abusive or harassing content. The scenario's goal of detecting harmful generated text includes identifying text that facilitates harassment or abuse.

Policy 3.a

"Intentionally deceive or mislead others, including use of Llama 2 related to the following:
- Generating, promoting, or furthering fraud or the creation or promotion of disinformation"

- This policy supports Generated Text Detection by addressing the misuse of LLMs to generate misleading or fraudulent content. Detecting such generated text aligns with the policy's goal of preventing deception and misinformation.

Policy 3.e

"Representing that the use of Llama 2 or outputs are human-generated."

- This policy underlines the importance of transparency in content generation. Generated Text Detection's goal is to determine whether a text is generated by an LLM or its fine-tuned version aligns with ensuring users do not misrepresent AI-generated content as human-generated.

In summary, The Llama 2 use policy explicitly prohibits various activities that relate to both scenarios defined in our paper, particularly IP infringement and the generation of illegal or harmful content.

F.2 Command R Series

Cohere R series is built on the language of business and is optimized for enterprise generative AI, search and discovery, and advanced retrieval. Their Cohere For AI Acceptable Use Policy[3] aligns with our scenario settings.

[3] https://docs.cohere.com/docs/c4ai-acceptable-use-policy

Scenario 1: Detecting IP Infringement

> Cohere For AI Acceptable Use Policy
>
> "Synthetic data for commercial uses: generating synthetic data outputs for commercial purposes, including to train, improve, benchmark, enhance or otherwise develop model derivatives, or any products or services in connection with the foregoing."

- This policy is highly relevant to IP Detection, as it explicitly prohibits using models or their derivatives for commercial purposes without permission, which is a core concern of detecting IP infringement.

Scenario 2: Detecting Generated Text

> Cohere For AI Acceptable Use Policy
>
> "We expect users of our models or model derivatives to comply with all applicable local and international laws and regulations. Additionally, you may not use or allow others to use our models or model derivatives in connection with any of the following strictly prohibited use cases:"

- This policy establishes the baseline expectation that users must comply with all laws and regulations, which supports the detection of misuse in both scenarios. Generated Text Detection specifically aligns with preventing the generation and dissemination of illegal content.

> Cohere For AI Acceptable Use Policy
>
> "Harassment and abuse: engaging in, promoting, facilitating, or inciting activities that harass or abuse individuals or groups."

- This policy supports Generated Text Detection by setting clear boundaries against generating content that could harass or abuse individuals or groups, aligning with the scenario's goals of detecting unethical content.

> Cohere For AI Acceptable Use Policy
>
> "Violence and harm: engaging in, promoting, or inciting violence, threats, hate speech self-harm, sexual exploitation, or targeting of individuals based on protected characteristics."

- This policy directly relates to Generated Text Detection, where the goal is to detect generated content that disseminates illegal, harmful, or unethical content. It provides a clear mandate against such misuse.

> Cohere For AI Acceptable Use Policy
>
> "Fraud and deception: misrepresenting generated content from models as human-created or allowing individuals to create false identities for malicious purposes, deception, or to cause harm, through methods including:
> - propagation of spam, fraudulent activities such as catfishing, phishing, or generation of false reviews;
> - creation or promotion of false representations of or defamatory content about real people, such as deepfakes; or
> - creation or promotion of intentionally false claims or misinformation."

- This is pertinent to both scenarios. For IP Detection, it addresses the misrepresentation of generated content as human-created, which can involve claiming an open-source LLM as a proprietary creation. For Generated Text Detection, it covers the generation of harmful or deceptive content.

F.3 01.ai Yi Series

The Yi series is another open-source LLM that has demonstrated excellent performance on the LMSYS Chatbot Arena Leaderboard[4]. This series of LLMs has been developed by a Chinese company named "Lingyiwanwu." The following user agreement[5] contains sections that align with the assumptions in our scenarios.

Scenario 1: Detecting IP Infringement

> **Article 5 Clause 1**
>
> "Lingyiwanwu is the developer and operator of this product and enjoys all rights to the data, information, and outputs generated during the development and operation of this product within the scope permitted by laws and regulations, except where the relevant rights holders are entitled to rights according to law."

- This clause asserts that Lingyiwanwu holds the rights to the outputs generated by the product, reinforcing the need to detect IP infringement when these rights are violated by unauthorized use or copying of the LLMs.

> **Article 5 Clause 2**
>
> "Unless otherwise agreed or stipulated by laws and regulations, you have the rights to the content generated based on the content you are entitled to upload and the rights to the content generated based on the uploaded content."

- This clause delineates user rights to generated content, provided it is based on legally uploaded content, highlighting the importance of detecting if generated content infringes on existing IP rights.

> **Article 5 Clause 6**
>
> "You understand and promise that your input during the use of this product will not infringe on any person's intellectual property rights, portrait rights, reputation rights, honor rights, name rights, privacy rights, personal information rights, etc. Otherwise, you will bear the risk and responsibility of infringement."

- This clause ensures that users acknowledge their responsibility to avoid infringing on IP rights, aligning with the scenario's assumption that detection mechanisms are needed to prevent such infringements.

> **Article 5 Clause 7**
>
> "If you add new data for model training, fine-tuning, and development during the use of this product, you will bear the resulting responsibilities."

- This clause emphasizes user responsibility for any new data added for model training or fine-tuning, aligning with the scenario's focus on detecting whether the generated text has been modified or fine-tuned from the original LLM.

[4] https://arena.lmsys.org/
[5] https://platform.lingyiwanwu.com/useragreement

Scenario 2: Detecting Generated Text

Article 4 Clause 1

"Based on your use of this product, Lingyiwanwu grants you a revocable, non-transferable, non-exclusive right to use this product. If you publish or disseminate content generated by this product, you should:

- Proactively verify the authenticity and accuracy of the output content to avoid spreading false information;
- Mark the output content as AI-generated in a prominent way to inform the public about the content synthesis;
- Avoid publishing and disseminating any output content that violates the usage norms of this agreement."

- This clause mandates users to verify and label AI-generated content, ensuring transparency and preventing the misuse of generated text for harmful or illegal purposes, which aligns with the scenario's goal of detecting and managing generated content responsibly.

Article 4 Clause 4

"Users are prohibited from engaging in certain behaviors, including but not limited to:

- **(5)** Inducing the generation of content that violates relevant laws and regulations or contains unfriendly outputs;
- **(7)** Developing products and services that compete with this product using this product;
- **(9)** Unauthorized removal or alteration of AI-generated labels or deep synthesis content labels."

- These prohibitions directly support the scenario's assumptions by preventing the generation and dissemination of harmful content, ensuring ethical use of the model, and maintaining the integrity of AI-generated labels for accountability.