# PRO: ENABLING PRECISE AND ROBUST TEXT WATERMARK FOR OPEN-SOURCE LLMS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Text watermarking for large language models (LLMs) is important for model owners to verify the origin and protect the intellectual property of AI-generated text. While watermarking methods for closed-source LLMs' text generation are relatively mature, watermarking open-source LLMs' text generation remains challenging. Closed-source model developers typically embed text watermarks during decoding; however, this approach is ineffective for the text generation of open-source models, where developers have no control over how decoding occurs. As a result, owners of open-source LLMs still lack practical methods to verify whether a given piece of AI-generated text originated from their models. The primary challenge lies in embedding watermarks directly into model weights without compromising detection accuracy. One possible solution is first to create a text generation watermark in the closed-source setting, then distill that watermark information into the publicly released model's weights. However, this approach faces two critical challenges: (i) Reduced detectability due to inconsistency between the watermark patterns learned by the model and the predefined patterns used during detection. This inconsistency arises because existing closed-source watermark patterns are difficult for models to learn effectively. (ii) Vulnerability to modifications by downstream users, such as fine-tuning or model merging, which may weaken or completely remove the embedded watermark. To address these challenges, we propose **PRO**, a precise and robust text watermarking method for open-source LLMs. First, we introduce a trainable watermark policy model, which is jointly optimized with the LLM during training. This co-optimization helps generate watermark patterns that are easier for the model to learn, significantly reducing inconsistencies between generated patterns and predefined detection criteria. Additionally, we incorporate a regularization term into the watermarking loss, which simulates various perturbations (e.g., fine-tuning, model merging) and penalizes any degradation in watermark detectability under these modifications. This approach ensures that the embedded watermark remains resilient even after downstream model alterations. Our evaluation on mainstream open-source LLMs (e.g., LLaMA-3.2, LLaMA-3, and Phi-2) demonstrates that our approach significantly outperforms prior methods in terms of both watermark detectability and robustness against model modifications. The code is publicly available at https://anonymous.4open.science/r/PRO.

## 1 INTRODUCTION

With the rapid advancement of LLMs and their widespread deployment, researchers and regulators have raised growing concerns regarding their potential misuse in generating harmful content (Bommasani et al., 2021; Union, 2021). To address this issue, text watermarking has emerged as a promising technique that embeds a watermark signal during text generation to facilitate the detection of LLM-generated content. Mainstream approaches (Kuditipudi et al., 2023; Kirchenbauer et al., 2023a; Hu et al., 2023) leverage a watermark scheme $f_w$ to generate watermark logits that bias the decoding process. During detection, the statistical distribution of tokens in the text is analyzed using $f_w$ to determine if it has been watermarked. As illustrated in Figure 1 (a), these decoding-based methods assume a closed-source LLM setting, where the LLM owner controls the entire inference pipeline, including the integration of $f_w$ into the decoding process.

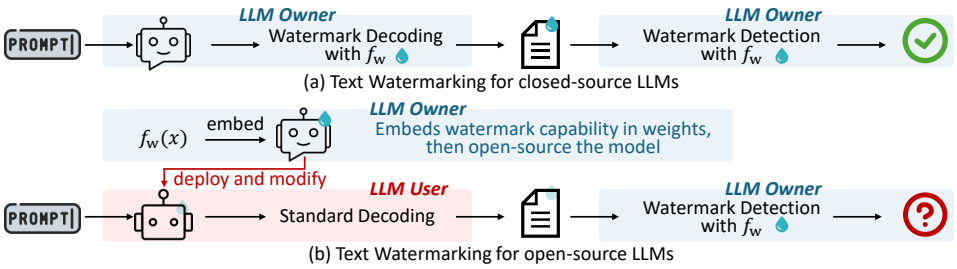


Figure 1: Text watermarking for (a) closed-source and (b) open-source LLMs. Closed-source watermarking relies on watermark decoding, while open-source watermarking requires embedding the watermark into the model weights so that standard decoding still produces watermarked text.

As open-source LLMs rapidly improve, the performance gap between open-source and closed-source LLMs is narrowing. Notably, some open-source LLMs such as DeepSeek (Liu et al., 2024b) and LLaMA-4 (Meta, 2024) are now matching or even surpassing closed-source LLMs like OpenAI's GPT-4 and Claude 3.5 on specific benchmarks (Guo et al., 2024). This underscores an urgent need for effective text watermarking for open-source LLMs. However, decoding-based methods are fundamentally unsuitable in the open-source setting, where LLM users will have full access to the inference pipeline to remove the watermark decoding. As illustrated in Figure 1 (b), a viable alternative involves embedding watermarking capability directly into the LLM's weights, enabling the LLM to generate watermarked texts naturally without relying on watermark decoding. In this context, two key challenges arise: (i) the *detectability* of watermarks in generated text, and (ii) the *robustness* against users' modifications on LLM's weights.

To integrate watermarking mechanisms directly into LLM weights, Christ et al. (2024) proposed shifting the addition of watermark logits from the decoding phase to a direct modification of the bias terms in the final projection layer. However, since prominent open-source LLMs typically omit bias terms in this layer, this requires architectural modifications that users could easily detect and remove. An alternative and promising direction, which avoids such architectural alterations, is to train the LLM on watermarked text (Gu et al., 2023; Sander et al., 2024). The objective here is for the LLM to natively learn the statistical patterns of the watermark, thereby generating watermarked logits as an inherent part of its output.

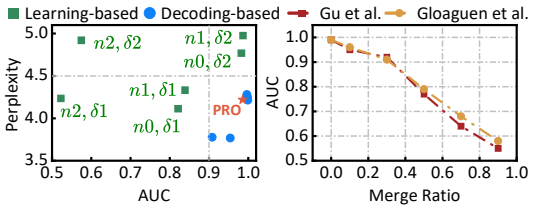


Figure 2: *(Left)* learning-based and decoding-based KGW watermarks under varying watermark hyperparameters $n$ and $\delta$. *(Right)* existing learning-based watermarks' AUC after merging with the unwatermarked LLM.

Despite its promise, existing learning-based watermarking faces critical challenges in *learnability* and *robustness*. First, unlike decoding-based watermarking, where the watermark logits are manually injected during the decoding process, learning-based methods require the LLM to learn to generate watermarked text directly. Its detectability is therefore highly contingent on the learnability of watermark signals, typically requiring a large watermark logits magnitude $\delta$ during training. However, training on highly distorted text impairs the LLM's generation quality. As shown in Figure 2 (left), learning-based watermark (■) trained with $\delta = 1$ only achieves $0.84$ AUC, while achieving $0.99$ AUC with $\delta = 2$ results in significant degradation of generation quality, i.e., Perplexity (PPL) increases from $4.1$ to $5.0$. This contrasts with decoding-based methods (●) that achieve both high quality (PPL $< 4.5$) and strong detectability (AUC $> 0.9$) simultaneously. Additionally, current learning-based methods are limited to small $n$-gram lengths ($n \leq 1$) (Gu et al., 2023; Sander et al., 2024; Zhao et al., 2025), as larger $n$ values significantly increase the complexity of watermark patterns for LLMs to learn, thereby diminishing watermark learnability. Even with $\delta = 2$, the LLMs fail to learn 2-gram KGW watermarks (AUC $= 0.54$). However, practical applications require higher $n$-gram (e.g., 3- or 4-grams) for robustness against reverse engineering (Kirchenbauer et al., 2023a; Zhao et al., 2025). Furthermore, the open-source setting allows users to modify LLMs through techniques like fine-tuning or model merging, which can inadvertently or intentionally remove learned watermarks. As shown in Figure 2 (right), merging a watermarked LLM with an

unwatermarked one at a 0.5 ratio reduces the watermark detection AUC to $0.79$. In contrast, the proposed PRO improves robustness, achieving an AUC of $0.87$, as reported in Table1 in Section 4.

In this paper, we propose **PRO** (**P**recise and **R**obust **O**pen-source LLM Watermark). Our journey begins by analyzing the aforementioned issues of existing watermarks for open-source LLMs.

First, current open-source LLM watermarking methods (Gu et al., 2023; Sander et al., 2024) distill predefined watermark patterns into model weights and use the same patterns for detection. This creates a *Generation-Detection Inconsistency*: the LLM often learns a deviated version of the watermark, while detection assumes the original pattern. This inconsistency stems from simply adopting watermarking methods like KGW (Kirchenbauer et al., 2023a) that were originally built for closed-source LLMs and were not intentionally designed to be learnable. These methods use predefined mapping functions to generate watermark logits. From the LLM's perspective, these mappings often appear arbitrary, forcing the LLM to memorize fragmented associations between prefixes and watermark logits rather than internalizing a coherent pattern of watermark logits. This challenge becomes catastrophic for large $n$-gram watermarks, where the complexity of mapping functions renders learning difficult. To resolve this, **PRO** introduces a trainable watermark policy that dynamically optimizes the watermark mapping function through joint training with the LLM. This co-adaptation ensures the policy generates watermark patterns that can be effectively learned by LLMs. Crucially, during detection, **PRO** employs the optimized policy instead of the predefined one, ensuring alignment with the watermark patterns the LLM has actually learned.

Second, to enhance robustness against user's weight modifications, Gloaguen et al. (2025) proposed embedding watermarks into "stable" parameters identified by observing value changes after fine-tuning on natural text. However, as shown in Figure 2 (right), this approach offers only marginal robustness against fine-tuning, since parameter stability is inherently dataset-dependent, and modifications like model merging or fine-tuning on other datasets can still erase watermarks. To address this, we propose the concept of *forgotten perturbation*: perturbations to weights that maximally degrade watermark detectability. To attenuate its negative impact, **PRO** iteratively generates perturbations using anti-watermarked text (adversarially crafted to erase watermarks), simulating powerful *forgotten perturbation*. During training, the LLM learns to withstand the *forgotten perturbation* by minimizing its disruptive effects while maintaining watermark detectability. By unifying perturbation resistance with watermark training in a single optimization framework, **PRO** achieves robustness against diverse user modifications.

We evaluate **PRO** on three mainstream open-source LLMs, including LLaMA3-8B, LLaMA3.2-3B and Phi2-2.7B. We embed watermarks and assess their robustness under common user modifications, such as quantization, pruning, fine-tuning, and model merging. Results reveal that **PRO** can achieve high detectability, low quality degradation and large $n$-gram lengths (i.e., $n \geq 5$), as shown in Figure 2 (left). These results represent substantial improvements over state-of-the-art open-source watermarking (Gu et al., 2023; Gloaguen et al., 2025) and can even match the performance of closed-source counterpart. Additionally, watermarks embedded by **PRO** are more robust against user modifications. **PRO** consistently preserves high detectability (AUC $\geq 0.80$) under aggressive model modifications, including high-ratio merging and long-step fine-tuning, marking the first precise and robust text watermark for open-source LLMs.

## 2 Related Work and Threat Model

### 2.1 Text watermarks for Closed-source LLMs

To ensure traceability of content generated by LLMs, watermarking techniques have been proposed to embed identifiable statistical signals into model outputs. Among these, *decoding-based watermarking* (Kirchenbauer et al., 2023a; Christ et al., 2024; Zhao et al., 2023; Kirchenbauer et al., 2023b) is a widely adopted approach. The watermark decoding function $f_w(\pi_\theta(x), \xi)$ leverages a secret key $\xi$ to transform the original next-token distribution $\pi_\theta(\cdot \mid x)$ into a modified distribution that embeds a detectable watermark signal in the generated text. This enables post-hoc detection via a test function $f_d(x, \xi)$, which computes a p-value indicating the presence of a watermark. However, decoding-based watermarking relies on customized decoding algorithms and is not applicable in open-model settings where users control the decoding process.

### 2.2 TEXT WATERMARKS FOR OPEN-SOURCE LLMS

Current watermarking schemes for open-source language models can be broadly categorized into two types: *Input Prompt-dependent* and *Input Prompt-independent*. The former requires access to the input prompt for detection, while the latter can detect watermarks using only the output text. Notably, our **PRO** falls into the category without requiring the input prompt.

#### 2.2.1 INPUT PROMPT-DEPENDENT

Several recent works propose open-source watermarking techniques that require input prompt for detection. Xu et al. (2024) jointly train a detector and an LLM to embed detectable signals into the output, requiring both the prompt and output for detection. Block et al. (2025) perturb model weights with Gaussian noise and detect watermarks via gradient-based statistical tests, which also rely on the input prompt. Both methods require access to the input prompt during detection, limiting their applicability in real-world scenarios.

#### 2.2.2 INPUT PROMPT-INDEPENDENT

We further divide input prompt-independent watermarking methods into two classes: learning-based watermarks and structural-editing watermarks.

*Learning-based Watermark.* Gu et al. (2023) shows that decoding-based watermarks can be embedded into model weights by distilling the watermark from a teacher LLM $\pi_o$. Then, the same watermark scheme can be used to detect the watermark in the student LLM $\pi_\theta$. The teacher generates watermarked data $\mathcal{D}_{\text{wm}}$, which the student fine-tunes using the cross-entropy loss:

$$\mathcal{L}_{\text{sampling}}(\pi_\theta) = \mathbb{E}_{x\sim\mathcal{D}_{\text{wm}}}\left[\sum_{t=1}^{|x|} -\log \pi_\theta(x_t|x_{<t})\right] \tag{1}$$

Alternatively, by exploiting the logits, the student can be fine-tuned to mimic the teacher LLM's next-token distribution using the KL-divergence loss:

$$\mathcal{L}_{\text{logit}}(\pi_\theta) = \mathbb{E}_{x\sim\mathcal{D}}\left[\sum_{t=1}^{|x|} \text{KL}(f_w(\pi_o(\cdot\,|\,x_{<t}), \xi_w) \parallel \pi_\theta(\cdot\,|\,x_{<t}))\right] \tag{2}$$

*Structural-editing Watermarks.* Christ et al. (2024) embed watermarks by adding small, token-specific Gaussian perturbations to the output-layer bias vector of the model. To detect whether a text sequence is watermarked, the LLM owner aggregates the bias perturbations of each output token. As most open-source LLMs disable output-layer biases by default[1], implementing this method requires explicitly enabling the bias term, which introduces an architectural modification. Such changes remain identifiable and can be easily removed by analyzing and modifying the architecture of the model.

### 2.3 THREAT MODEL

We consider a threat model where an open-source LLM is publicly released to users, who have full access to the model's weights and architecture. Users may modify the model through fine-tuning, model merging, quantization, or pruning. The LLM owner embeds a watermark into the released model's weights and retains a private detection mechanism, which may be made available via a detection API. Detection is performed solely on the generated text, without access to the user's input prompt or control over the decoding process.

## 3 METHODS

### 3.1 OVERVIEW

Learning-based watermarking aims to train LLMs to generate watermarked text natively, without modifying the decoding scheme. Specifically, given an original LLM $\pi_o$ and a watermarked decoding

[1]For example, the output layers of LLaMA, Qwen, and Mistral models in the Hugging Face Transformers library are defined with `bias=False`.

function $f_w$, the combination acts as a teacher model. The objective is to train a student LLM $\pi_\theta$ with standard decoding such that its next-token distribution $\pi_\theta(x)$ approximates the teacher's output, $f_w(\pi_o(x), \xi)$, for any input $x$. Our goal is to build a precise and robust learning-based watermark for open-source LLMs that (i) maintains high detection accuracy without degrading generation quality, and (ii) remains robust against general user modifications such as fine-tuning. We first discuss the core challenges in achieving these goals and then present how our proposed **PRO** addresses them.

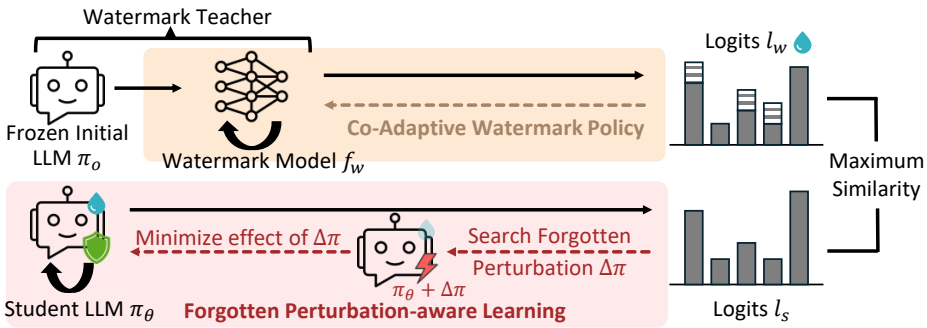


Figure 3: Overview of our proposed method. CAWP (*upper*) jointly trains a watermark model with the student LLM to generate learning-friendly watermark logits. FPL (*bottom*) improves robustness by searching and minimizing the effect of forgotten perturbations that may erase the watermark.

### 3.2 Co-Adaptive Watermark Policy (CAWP)

Watermark detection for open-source LLMs involves two steps: (1) the LLM user generates text via the watermarked LLM, and (2) the LLM owner uses the predefined watermark pattern to detect the text. In this case, an important consideration is the *consistency* between the watermark pattern learned by the LLM and the predefined one used during detection. To assess this *consistency*, we fine-tune a LLaMA3-8B on text generated by a decoding-based watermarked LLaMA3-8B (i.e., watermark teacher), thereby obtaining a learning-based watermarked LLaMA3-8B (i.e., watermark student). We then measure and compare the green token ratios across them in Figure 4. The results reveal a significant green ratio drift in the student relative to the teacher, which reduces the AUC from $0.99$ to $0.84$, indicating that the student LLM fails to fully internalize the watermarking pattern of the watermark teacher. To mitigate the inconsistency, we identify two primary optimization directions. The first is to design a learning-friendly watermark pattern, thereby enhancing its inherent learnability for the student LLM. The second is to adapt the detection mechanism itself, using a pattern that aligns more closely with the watermark distribution actually learned by the student LLM, rather than strictly adhering to the original predefined pattern.

Unlike prior work that uses rigid, predefined watermark functions, we introduce Co-Adaptive Watermark Policy (CAWP) as illustrated in Figure 3 (upper). It co-optimizes a trainable watermark policy model with the student LLM, allowing the watermark patterns to adapt to the LLM's learning dynamics and become more learnable. Crucially, detection leverages the co-optimized patterns rather than the original predefined ones, ensuring detection aligns with the watermark signals internalized by the LLM to mitigate generation-detection inconsistency.

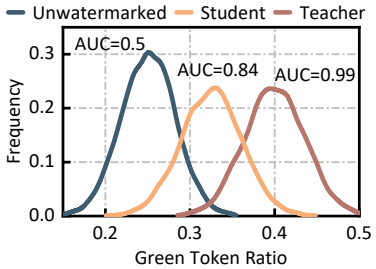


Figure 4: Green token ratios for unwatermarked, watermark student, and watermark teacher (KGW with green ratio $= 0.25$, $n = 1$, $\delta = 1$).

The watermark policy consists of a pre-trained Embedding Model $E$ (e.g., BERT) and a trainable Watermark Mapping Model $M$ (e.g., an MLP). Given a sequence of preceding tokens $\{x_{i-n:i-1}\}$ at position $i$, $E$ generates their embeddings $\{e_{i-n:i-1}\}$. These embeddings are then transformed by $M$ into raw watermark logits over the vocabulary, i.e., $M(E(x_{i-n:i-1}))$. The final watermark logits are obtained by scaling with a strength coefficient $\delta$, yielding $\delta \cdot M(E(x_{i-n:i-1}))$, which are added to the next-token logits before sampling. Given a dataset of texts $\mathcal{D}$, the training objective is to minimize

the mean KL divergence between teacher and student next token distributions on all prefixes in $\mathcal{D}$:

$$\mathcal{L}_{\text{sim}} = \frac{1}{|\mathcal{D}|} \sum_{x \in \mathcal{D}} \sum_{i=n}^{N-1} \text{KL}\left[ \overbrace{\pi_o(\cdot \mid x_{\leq i}) + \underbrace{\delta \cdot M(E(x_{i-n:i-1}))}_{\text{watermark logits}}}^{\text{teacher logits}} \middle\| \overbrace{\pi_\theta(\cdot \mid x_{\leq i})}^{\text{student logits}} \right] \tag{3}$$

Here, $n$ is the gram length. Both the teacher LLM $\pi_o$ and the embedding model $E$ are frozen; only the student model $\pi_\theta$ and mapping model $M$ are optimized. The student LLM $\pi_\theta$ is optimized by minimizing $\mathcal{L}_{\text{sim}}$, while the mapping model $M$ must also satisfy the following requirements: (i) *Unbiased Token Preference*: the watermark logits should not exhibit persistent positive or negative bias toward specific tokens. (ii) *Balanced Watermark Logits*: the logits should have zero mean across the vocabulary, ensuring symmetric perturbation that makes approximately half the tokens more likely and the other half less likely. (iii) *Non-Vanishing Watermark Logits*: to avoid degenerate solutions where the watermark logits vanish, each watermark logit is encouraged to maintain a minimum absolute magnitude, regularized toward a target value $\epsilon$.

$$\mathcal{L}_{\text{norm}} = \overbrace{\sum_i \left| \frac{1}{|\mathcal{V}|} \sum_j M(e_i)^{(j)} \right|}^{\text{Unbiased Token Preference}} + \overbrace{\sum_j \left| \frac{1}{N} \sum_i M(e_i)^{(j)} \right|}^{\text{Balanced Watermark Logits}} + \lambda_1 \overbrace{\sum_{i,j} \max\left(0, \epsilon - \left| M(e_i)^{(j)} \right| \right)}^{\text{Non-Vanishing Watermark Logits}} \tag{4}$$

$M(e_i)^{(j)}$ denotes the watermark logit for the $j$-th token in the vocabulary (of size $|\mathcal{V}|$) corresponding to the $i$-th input embedding $e_i$, with $N$ total input samples. The index $i$ sums over all samples, and $j$ sums over all vocabulary tokens. The final training loss for the watermark mapping model $M$ is the weighted sum of the similarity loss and the normalization loss, given by:

$$\mathcal{L}_M = \mathcal{L}_{\text{sim}} + \lambda_2 \mathcal{L}_{\text{norm}} \tag{5}$$

Upon training, the watermarked student LLM $\pi_\theta$ is released, and the co-optimized watermark mapping model $M$ is retained for detection. For a given text, the LLM owner computes the watermark logit at each position $i$ by first embedding the $n$-gram prefix with $E$, then transforming it via the mapping model $M$ to obtain the watermark logit for the actual next token $x_i$. The detection score is the average watermark logit across the sequence, given by $z = {}^1\!/_N \sum_{i=1}^{N} M\big(E(x_{i-n:i-1})\big)^{(x_i)}$. If $z$ exceeds a predefined threshold, the text is considered watermarked.

While some prior work also employs neural networks to generate watermark logits, they target different goals. For example, Liu et al. (2024a) and Ren et al. (2023) design semantic-invariant watermark models to improve robustness against paraphrasing attacks. In contrast, our CAWP framework is fundamentally different. Existing approaches are designed for closed-source LLMs and do not consider the learnability of the watermark by the model itself—their watermark models are trained independently from the LLM. By contrast, CAWP focuses on generating *learning-friendly* watermark logits by jointly optimizing the watermark model with the student LLM being watermarked. Additionally, such joint training ensures a bidirectional alignment: the watermark pattern used for detection adapts toward the student LLM's learned representation, while the student LLM is simultaneously guided to internalize patterns consistent with detection. This mutual convergence narrows the gap between the student and teacher distributions in Figure 4.

### 3.3 Forgotten Perturbation-aware Learning (FPL)

The vulnerability of watermarked LLMs to user modifications arises primarily from *forgotten perturbation*. Specifically, when a user modifies a watermarked LLM $\pi_\theta$ into $\pi'_\theta$, the weight drift $\Delta\pi_\theta = \pi'_\theta - \pi_\theta$ may remove the learned watermark. We term the drift that erases the watermark as *forgotten perturbation*, and our goal is to minimize its effect during the watermark learning stage.

To understand its root cause, we investigate fine-tuning as a representative user modification, i.e., $\pi_{t+1} = \pi_t - \eta\, g(\pi_t)$, where $g(\pi_t)$ is the gradient on user data driving weight drift. Figure 5 shows that fine-tuning a watermarked LLaMA3-8B on raw texts (a mixture of green/red tokens) collapses detectability, whereas fine-tuning on watermarked texts (green-token dominant) preserves it. This indicates

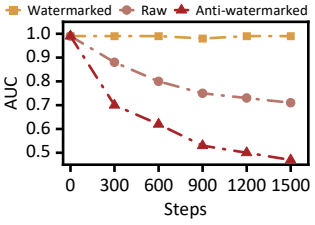


Figure 5: AUC during fine-tuning on raw, watermarked, and anti-watermarked texts.

that gradients arising from red tokens are the primary cause of *forgotten perturbation*. To further validate this, we fine-tune on *anti-watermarked* texts, generated with inverted watermark logits to make texts red-token dominant. As shown in Figure 5, fine-tuning on these anti-watermarked texts results in a substantial drop in detectability. These results confirm that red-token-driven *forgotten perturbations* are the dominant factor in watermark forgetting, and mitigating their impact should be a key objective.

To achieve this, we propose Forgotten Perturbation-aware Learning (FPL), which explicitly reduces model sensitivity to *forgotten perturbation*, as illustrated in Figure 3 (bottom). Our objective is to train a watermarked model that can not only generate watermarked texts but also attenuate the effect of future perturbations caused by red-token updates. The training objective of the LLM is defined as:

$$\arg\min_{\boldsymbol{\pi}_\theta} \mathcal{L}_{\text{sim}}(\boldsymbol{\pi}_\theta) + \beta \left( \mathcal{L}_{\text{anti}}(\boldsymbol{\pi}_\theta) - \mathcal{L}_{\text{anti}}(\boldsymbol{\pi}_\theta - \alpha \frac{\nabla \mathcal{L}_{\text{anti}}(\boldsymbol{\pi}_\theta)}{\|\nabla \mathcal{L}_{\text{anti}}(\boldsymbol{\pi}_\theta)\|}) \right) \tag{6}$$

Here, $\mathcal{L}_{\text{sim}}$ is the watermark learning loss over watermarked texts (as defined in Equation 3), and $\mathcal{L}_{\text{anti}}$ is the watermark forgetting loss evaluated on anti-watermarked texts, defined analogous to $\mathcal{L}_{\text{sim}}$ using KL divergence loss. The second term measures the change in forgetting loss before and after applying a normalized step in the direction of the *forgotten perturbation*. The regularization weight $\beta$ controls the trade-off, and $\alpha$ is the perturbation step size. This formulation encourages the model to not only learn a strong watermark but also be robust against potential forgetting induced by user-side modifications.

Although the optimization in Equation 6 involves second-order derivatives, we can solve it using three forward/backward passes: (1) evaluate $\mathcal{L}_{\text{sim}}(\pi_\theta)$ on watermarked logits, (2) compute $\mathcal{L}_{\text{anti}}(\pi_\theta)$ on anti-watermarked data and perform a backward pass to obtain $\nabla \mathcal{L}_{\text{anti}}(\pi_\theta)$, and (3) simulate one normalized fine-tuning step along the forgetting gradient and perform another forward pass to compute $\mathcal{L}_{\text{anti}}(\pi_\theta - \alpha \hat{g})$, where $\hat{g} = \nabla \mathcal{L}_{\text{anti}} / \|\nabla \mathcal{L}_{\text{anti}}\|$. The difference between the pre- and post-step losses reflects the watermarked LLM's vulnerability to *forgotten perturbation*, which we aim to minimize. It is worth noting that Equation 6 is used only for training the LLM $\boldsymbol{\pi}_\theta$, while the watermark mapping model $M$ is still optimized with the loss in Equation 5. For completeness, the formal convergence analysis of CAWP is presented in the Appendix D, along with the detailed mathematical derivation of the second-order derivative term in Equation 6.

# 4 Experiments

## 4.1 Experimental settings

**Models.** We perform experiments on three open-source LLMs: LLaMA3-8B, Phi2-2.7B, and LLaMA3.2-3B. These models cover a range of model sizes from lightweight to large-scale models. The watermark mapping model $M$ is a lightweight MLP: a linear projection $\mathbb{R}^{1024} \to \mathbb{R}^{500}$, two ReLU-based residual blocks, and a final projection to the vocabulary dimension with $\tanh$ to constrain logits to $[-1, 1]$.

**Implementation Details.** In our watermarking pipeline, the semantic embeddings are generated by `compositional-bert-large` (Chanchani & Huang, 2023). During training, both the student LLM $\pi_\theta$ and the watermark mapping model $M$ are jointly optimized to maximize learnability and detection consistency. We set the regularization weights $\lambda_1 = \lambda_2 = 1$ in the training loss for CAWP. For the FPL component, we use perturbation step size $\alpha = 0.1$ and regularization weight $\beta = 5$. In our ablation study, we further analyze the sensitivity of watermark robustness to variations in these hyperparameters.

**Baseline Watermarking Methods.** Our experiments include three baseline watermarking methods, all implemented via KL-based distillation from a decoding-based teacher model. The first is Gu et al.-KGW. Following the setup in (Gu et al., 2023), we distill KGW with a fixed $\gamma = 0.25$ and evaluate three configurations of $(k, \delta)$: $(1, 2)$, $(0, 1)$, and $(1, 1)$. The second baseline is Gu et al.-KTH, which distills the exponential decoding watermarking scheme proposed in (Kuditipudi et al., 2023). The third is Gloaguen et al.-KGW, which builds on KGW by embedding watermarks into "stable" parameters identified via contrastive task vector analysis before and after fine-tuning on raw data. The details of training configurations and device usage can be found in Appendix H.3.

**Metrics.** We evaluate open-source watermarking methods along three dimensions: detectability, text quality, and robustness. For detectability, we follow (Gu et al., 2023), where each watermarked model generates 5,000 samples by prompting with 50-token prefixes from the C4 dataset (Dodge et al., 2021) and generating 200-token continuations. Standard sampling with temperature 1 is used during generation to ensure consistency across methods. Detection is performed by comparing these generations against 5,000 non-watermarked texts using the corresponding watermark detection algorithm, and we report the Area Under the ROC Curve (AUC). Text quality is measured via median perplexity (PPL), using a LLaMA-2-13B model. To assess robustness, we apply four types of model modifications that simulate real-world user behavior, modification settings are in Appendix H.3.

## 4.2 Results

### 4.2.1 Comparison with Existing Works

**Detectability and Quality Analysis.** To demonstrate **PRO**'s effectiveness, we compare it against existing text watermarking methods for open-source LLMs, including (Gu et al., 2023) and (Gloaguen et al., 2025). To assess the tradeoff between detectability and generation quality, we vary the watermark hyperparameters ($\delta$ and $n$-gram for KGW, $s$ for KTH, $\delta$ for **PRO**) and measure perplexity and detection AUC. The hyperparameter configuration can be found in the Appendix H.1. As shown in Figure 6, existing open-source watermarks exhibit quality degradation, while **PRO** shows the lowest effect on the quality of text. **PRO** bridges this gap: by co-optimizing a dynamic watermark policy with the watermarked LLM, it discovers learning-friendly watermark patterns that LLMs can internalize under low-distortion conditions. This achieves an AUC of 0.99 while reducing perplexity by 20.5% compared to the best baseline across all tested LLMs. ROC curves are shown in Figure 7.

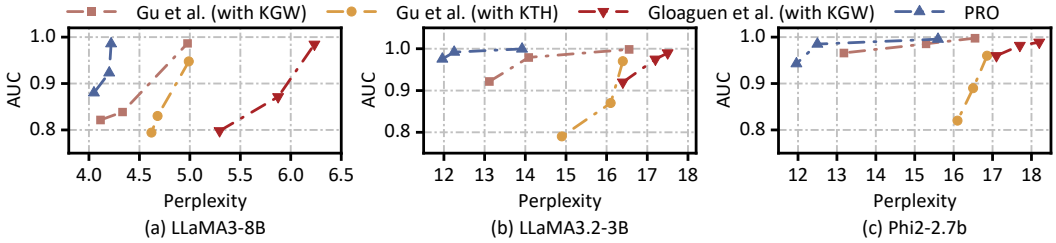


Figure 6: Effectiveness comparison of different open-source LLMs text watermarks, in terms of detection AUC and median PPL on three LLMs. A better watermark detectability and generation quality is indicated by higher AUC and lower PPL, as shown by lines closer to the upper left corner.

**Robustness against Model Modifications.** We evaluate the robustness of **PRO** under common model modification scenarios, including quantization, pruning, model merging, and fine-tuning. As shown in Table 1, existing watermarking methods generally perform reliably under *quantization* and *pruning*, where the parameters shifts to the model are limited and the impact on learned weights is relatively small. However, under *model merging* and *fine-tuning*, which induce more substantial and less predictable changes in LLM weights, the performance of existing methods degrades noticeably. For example, when merging a watermarked LLM with the original one at an interpolation ratio of $t = 0.5$, the AUC of (Gu et al., 2023) with KGW decreases to 0.773, while **PRO** retains a higher AUC of 0.876. For fine-tuning, after 1500 fine-tuning steps on the OpenMathInstruct dataset, the AUC of the (Gu et al., 2023) with KTH drops to 0.524, whereas **PRO** maintains 0.808. Consistent improvement is also shown on OpenCodeInstruct dataset. These results indicate that **PRO** exhibits improved resilience to more aggressive forms of model modifications. We attribute this improvement to the incorporation of FPL, which explicitly simulates and counteracts watermark forgetting via forgotten perturbations during training.

**Computational Efficiency.** Our method maintains comparable computational cost to prior watermarking approaches. CAWP introduces only a lightweight MLP watermark model (1.16M parameters, 0.0388% of a 3B LLM), which is negligible relative to the base model. FPL requires two additional forward/backward passes per iteration, but this overhead is offset by faster convergence: PRO reaches AUC 0.997 within 2000 steps, whereas KGW requires 5000 steps for a maximum AUC of 0.991. As a result, the overall wall-clock training time remains comparable to that of the baselines (see Appendix E for detailed analysis).

Table 1: AUC, TPR at 5 % FPR and PPL of different watermark methods of LLaMA-3-8B under model modifications. Colors indicate AUC: green for $\geq 0.8$, red for $< 0.7$, yellow otherwise.

| Category | Method | Config | Gu et al. (2023)-KTH | | | Gu et al. (2023)-KGW | | | Gloaguen et al. (2025)-KGW | | | **PRO** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | AUC | TPR@5 | PPL | AUC | TPR@5 | PPL | AUC | TPR@5 | PPL | AUC | TPR@5 | PPL |
| Unaltered | | | 0.951 | 0.779 | 5.0 | 0.991 | 0.956 | 4.9 | 0.990 | 0.943 | 6.2 | 0.997 | 0.990 | 4.2 |
| Quantization | 8 bits | GPTQ | 0.928 | 0.657 | 5.2 | 0.985 | 0.921 | 5.0 | 0.991 | 0.945 | 6.3 | 0.992 | 0.969 | 4.3 |
| | | INT8 | 0.934 | 0.694 | 5.1 | 0.980 | 0.907 | 5.0 | 0.990 | 0.941 | 6.4 | 0.984 | 0.931 | 4.4 |
| | 4 bits | HQQ | 0.882 | 0.535 | 6.1 | 0.992 | 0.954 | 5.5 | 0.983 | 0.911 | 7.9 | 0.995 | 0.981 | 4.8 |
| | | GPTQ | 0.915 | 0.635 | 5.5 | 0.982 | 0.910 | 5.1 | 0.988 | 0.935 | 7.3 | 0.987 | 0.942 | 5.1 |
| Pruning | Wanda | $\rho = 0.2$ | 0.941 | 0.728 | 8.6 | 0.992 | 0.959 | 7.5 | 0.995 | 0.974 | 9.9 | 0.997 | 0.988 | 7.0 |
| | | $\rho = 0.5$ | 0.921 | 0.639 | 8.4 | 0.973 | 0.860 | 6.9 | 0.987 | 0.934 | 8.9 | 0.990 | 0.957 | 6.2 |
| | SparseGPT | $\rho = 0.2$ | 0.955 | 0.783 | 7.7 | 0.990 | 0.948 | 8.1 | 0.990 | 0.952 | 9.0 | 0.995 | 0.983 | 7.1 |
| | | $\rho = 0.5$ | 0.951 | 0.755 | 8.1 | 0.981 | 0.906 | 7.8 | 0.986 | 0.928 | 9.6 | 0.993 | 0.971 | 8.0 |
| Merging | SLERP | $t = 0.1$ | 0.811 | 0.360 | 5.2 | 0.953 | 0.762 | 4.3 | 0.964 | 0.817 | 6.1 | 0.983 | 0.922 | 4.2 |
| | | $t = 0.3$ | 0.714 | 0.212 | 5.1 | 0.924 | 0.651 | 4.0 | 0.916 | 0.618 | 5.3 | 0.962 | 0.828 | 4.1 |
| | | $t = 0.5$ | 0.563 | 0.094 | 4.8 | 0.773 | 0.304 | 3.8 | 0.799 | 0.338 | 4.6 | 0.876 | 0.569 | 4.0 |
| | | $t = 0.7$ | 0.520 | 0.064 | 4.3 | 0.648 | 0.153 | 3.7 | 0.685 | 0.187 | 4.2 | 0.774 | 0.308 | 3.9 |
| | | $t = 0.9$ | 0.513 | 0.057 | 4.1 | 0.558 | 0.114 | 3.6 | 0.588 | 0.106 | 4.0 | 0.68 | 0.165 | 3.8 |
| Finetuning | OpenMath Instruct | $s = 300$ | 0.715 | 0.211 | 5.1 | 0.852 | 0.478 | 4.1 | 0.847 | 0.450 | 5.5 | 0.928 | 0.703 | 4.1 |
| | | $s = 600$ | 0.669 | 0.166 | 4.9 | 0.811 | 0.383 | 3.7 | 0.834 | 0.408 | 5.2 | 0.871 | 0.509 | 3.9 |
| | | $s = 900$ | 0.572 | 0.091 | 4.8 | 0.758 | 0.298 | 3.6 | 0.783 | 0.313 | 5.2 | 0.847 | 0.462 | 3.7 |
| | | $s = 1200$ | 0.532 | 0.089 | 4.7 | 0.731 | 0.250 | 3.5 | 0.739 | 0.249 | 5.0 | 0.818 | 0.403 | 3.6 |
| | | $s = 1500$ | 0.524 | 0.022 | 4.5 | 0.721 | 0.231 | 3.5 | 0.718 | 0.222 | 5.1 | 0.808 | 0.368 | 3.6 |
| | OpenCode Instruct | $s = 300$ | 0.768 | 0.274 | 7.3 | 0.924 | 0.634 | 6.9 | 0.926 | 0.653 | 7.4 | 0.929 | 0.707 | 7.0 |
| | | $s = 600$ | 0.722 | 0.224 | 5.7 | 0.868 | 0.453 | 4.9 | 0.874 | 0.477 | 7.0 | 0.886 | 0.558 | 5.4 |
| | | $s = 900$ | 0.679 | 0.163 | 4.8 | 0.805 | 0.323 | 5.0 | 0.813 | 0.344 | 6.8 | 0.856 | 0.477 | 5.1 |
| | | $s = 1200$ | 0.612 | 0.112 | 4.7 | 0.781 | 0.290 | 4.3 | 0.796 | 0.307 | 6.3 | 0.833 | 0.404 | 4.8 |
| | | $s = 1500$ | 0.533 | 0.071 | 4.6 | 0.754 | 0.259 | 4.2 | 0.782 | 0.304 | 5.9 | 0.821 | 0.391 | 4.0 |

### 4.2.2 Ablation Study

To understand the contributions of each component proposed, we conduct an ablation study by varying the perturbation step size $\alpha$ and the regularization weight $\beta$ in Equation 6. When both are set to zero, the model essentially reduces to using CAWP alone. In this case, robustness improves slightly over the baseline (Merge AUC $= 0.82$ after merging), indicating CAWP alone can already enhance watermark robustness. Introducing FPL via non-zero $\beta$ brings further gains in robustness (AUC $= 0.87$ at $\alpha = 0.1$, $\beta = 5$), though with a mild increase in perplexity. However, larger values of $\alpha$ or $\beta$ lead to degraded or unstable results, highlighting the importance of careful hyperparameter tuning to balance robustness and generation quality.

Table 2: Ablation on perturbation step size $\alpha$ and regularization weight $\beta$. We report AUC and PPL of unaltered model and AUC after merging ($t = 0.5$) to indicate robustness.

| $\alpha$ | 0 | **0.1** | 1 | 2 | 5 |
|---|---|---|---|---|---|
| AUC | 0.99 | **0.99** | 0.98 | 0.97 | 0.95 |
| PPL | 4.18 | **4.22** | 4.30 | 4.70 | 5.00 |
| Merge AUC | 0.82 | **0.87** | 0.86 | 0.83 | 0.79 |

(a) Effect of perturbation step size $\alpha$.

| $\beta$ | 0 | 1 | **5** | 10 |
|---|---|---|---|---|
| AUC | 0.99 | 1.00 | **0.99** | 0.95 |
| PPL | 4.18 | 4.21 | **4.22** | 4.52 |
| Merge AUC | 0.82 | 0.85 | **0.87** | 0.84 |

(b) Effect of regularization weight $\beta$.

### 4.2.3 Other Experiments

We provide additional results in the Appendix, including convergence analysis, computational efficiency, robustness under model modifications, and evaluations on extra datasets and models. We also report TPRs at multiple FPRs, robustness to paraphrasing, and comparisons with classifier-based detectors. These supplemental experiments further demonstrate that PRO generalizes well across models, datasets, and evaluation settings.

## 5 Conclusion

We identify the key challenges in watermarking open-source LLMs, the low learnability of predefined watermark patterns and their vulnerability to model modifications. To address these issues, we propose **PRO**, a precise and robust framework that jointly trains a learnable watermark policy with the LLM and incorporates perturbation-aware optimization to enhance robustness. **PRO** improves watermark detectability and resilience while maintaining generation quality, providing a practical solution for open-source LLM text watermark.

## REFERENCES


Wasi Uddin Ahmad, Aleksander Ficek, Mehrzad Samadi, Jocelyn Huang, Vahid Noroozi, Somshubra Majumdar, and Boris Ginsburg. Opencodeinstruct: A large-scale instruction tuning dataset for code llms. *arXiv preprint arXiv:2504.04030*, 2025.

Hicham Badri and Appu Shaji. Half-quadratic quantization of large machine learning models, November 2023. URL `https://mobiusml.github.io/hqq_blog/`.

Adam Block, Ayush Sekhari, and Alexander Rakhlin. Gaussmark: A practical approach for structural watermarking of language models. *arXiv preprint arXiv:2501.13941*, 2025.

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

Sachin J Chanchani and Ruihong Huang. Composition-contrastive learning for sentence embeddings. *arXiv preprint arXiv:2307.07380*, 2023.

Miranda Christ, Sam Gunn, Tal Malkin, and Mariana Raykova. Provably robust watermarks for open-source language models. *arXiv preprint arXiv:2410.18861*, 2024.

Rocktim Jyoti Das, Mingjie Sun, Liqun Ma, and Zhiqiang Shen. Beyond size: How gradients shape pruning decisions in large language models. *arXiv preprint arXiv:2311.04902*, 2023.

Tim Dettmers, Mike Lewis, Younes Belkada, and Luke Zettlemoyer. Gpt3. int8 (): 8-bit matrix multiplication for transformers at scale. *Advances in neural information processing systems*, 35: 30318–30332, 2022.

Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *Advances in neural information processing systems*, 36:10088–10115, 2023.

Jesse Dodge, Maarten Sap, Ana Marasović, William Agnew, Gabriel Ilharco, Dirk Groeneveld, Margaret Mitchell, and Matt Gardner. Documenting large webtext corpora: A case study on the colossal clean crawled corpus. *arXiv preprint arXiv:2104.08758*, 2021.

Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International conference on machine learning*, pp. 1126–1135. PMLR, 2017.

Elias Frantar and Dan Alistarh. Sparsegpt: Massive language models can be accurately pruned in one-shot. In *International Conference on Machine Learning*, pp. 10323–10337. PMLR, 2023.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Thibaud Gloaguen, Nikola Jovanović, Robin Staab, and Martin Vechev. Towards watermarking of open-source llms. *arXiv preprint arXiv:2502.10525*, 2025.

Charles Goddard, Shamane Siriwardhana, Malikeh Ehghaghi, Luke Meyers, Vladimir Karpukhin, Brian Benedict, Mark McQuade, and Jacob Solawetz. Arcee's mergekit: A toolkit for merging large language models. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pp. 477–485, 2024.

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

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

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

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

John Kirchenbauer, Jonas Geiping, Yuxin Wen, Manli Shu, Khalid Saifullah, Kezhi Kong, Kasun Fernando, Aniruddha Saha, Micah Goldblum, and Tom Goldstein. On the reliability of watermarks for large language models. *arXiv preprint arXiv:2306.04634*, 2023b.

Kalpesh Krishna, Yixiao Song, Marzena Karpinska, John Wieting, and Mohit Iyyer. Paraphrasing evades detectors of ai-generated text, but retrieval is an effective defense. *Advances in Neural Information Processing Systems*, 36:27469–27500, 2023.

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

Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. Awq: Activation-aware weight quantization for on-device llm compression and acceleration. *Proceedings of Machine Learning and Systems*, 6: 87–100, 2024.

Aiwei Liu, Leyi Pan, Xuming Hu, Shiao Meng, and Lijie Wen. A semantic invariant robust watermark for large language models. In *The Twelfth International Conference on Learning Representations*, 2024a. URL https://openreview.net/forum?id=6p8lpe4MNf.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024b.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. Llm-pruner: On the structural pruning of large language models. *Advances in neural information processing systems*, 36:21702–21720, 2023.

Meta. Llama-4. https://ai.meta.com/blog/llama-4-multimodal-intelligence/, 2024.

Aravind Rajeswaran, Chelsea Finn, Sham M Kakade, and Sergey Levine. Meta-learning with implicit gradients. *Advances in neural information processing systems*, 32, 2019.

Jie Ren, Han Xu, Yiding Liu, Yingqian Cui, Shuaiqiang Wang, Dawei Yin, and Jiliang Tang. A robust semantics-based watermark for large language model against paraphrasing. *arXiv preprint arXiv:2311.08721*, 2023.

Tom Sander, Pierre Fernandez, Alain Durmus, Matthijs Douze, and Teddy Furon. Watermarking makes language models radioactive. *Advances in Neural Information Processing Systems*, 37: 21079–21113, 2024.

Mingjie Sun, Zhuang Liu, Anna Bair, and J Zico Kolter. A simple and effective pruning approach for large language models. *arXiv preprint arXiv:2306.11695*, 2023.

Shubham Toshniwal, Ivan Moshkov, Sean Narenthiran, Daria Gitman, Fei Jia, and Igor Gitman. Openmathinstruct-1: A 1.8 million math instruction tuning dataset. *Advances in Neural Information Processing Systems*, 37:34737–34774, 2024.

E Union. Proposal for a regulation of the european parliament and of the council laying down harmonised rules on artificial intelligence (artificial intelligence act) and amending certain union legislative acts. *COM/2021/206final*, 2021.

Mengzhou Xia, Tianyu Gao, Zhiyuan Zeng, and Danqi Chen. Sheared llama: Accelerating language model pre-training via structured pruning. *arXiv preprint arXiv:2310.06694*, 2023.

Xiaojun Xu, Yuanshun Yao, and Yang Liu. Learning to watermark llm-generated text via reinforcement learning. *arXiv preprint arXiv:2403.10553*, 2024.

Enneng Yang, Li Shen, Guibing Guo, Xingwei Wang, Xiaochun Cao, Jie Zhang, and Dacheng Tao. Model merging in llms, mllms, and beyond: Methods, theories, applications and opportunities. *arXiv preprint arXiv:2408.07666*, 2024.

Biao Zhang, Zhongtao Liu, Colin Cherry, and Orhan Firat. When scaling meets llm finetuning: The effect of data, model and finetuning method. *arXiv preprint arXiv:2402.17193*, 2024.

Shengyu Zhang, Linfeng Dong, Xiaoya Li, Sen Zhang, Xiaofei Sun, Shuhe Wang, Jiwei Li, Runyi Hu, Tianwei Zhang, Fei Wu, et al. Instruction tuning for large language models: A survey. *arXiv preprint arXiv:2308.10792*, 2023.

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

Zhengyue Zhao, Xiaogeng Liu, Somesh Jha, Patrick McDaniel, Bo Li, and Chaowei Xiao. Can watermarks be used to detect llm ip infringement for free? In *The Thirteenth International Conference on Learning Representations*, 2025.

# APPENDIX

# A THE USE OF LARGE LANGUAGE MODELS (LLMS)

The authors used ChatGPT and Grammarly to check and correct any typos and grammatical errors.

# B ETHICS STATEMENT

This work focuses on developing robust watermarking techniques for open-source large language models to support provenance verification and responsible AI use. Our study does not involve human or animal subjects, nor does it require collection of personal or sensitive data. All experiments are conducted on publicly available pretrained models and benchmark datasets. We believe our method enhances transparency and accountability in AI deployment, and we are not aware of any ethical concerns or potential harms arising from this research.

# C REPRODUCIBILITY STATEMENT

We have taken multiple steps to ensure the reproducibility of our work. The code is publicly available at https://anonymous.4open.science/r/PRO, together with a README file that includes instructions for installation, configuration, and execution of experiments.

# D THEORETICAL ANALYSIS

## D.1 CONVERGENCE ANALYSIS OF CAWP

We anchor our proof in established convergence guarantees for distillation-based LLM training, where a student LLM $\pi_\theta$ converges to a fixed teacher $\pi_o$ under KL divergence minimization. The core innovation of CAWP is introducing a trainable watermark policy $M_\phi$ that perturbs the teacher distribution. We prove convergence by analyzing this perturbation.

### D.1.1 BASELINE DISTILLATION CONVERGENCE (FIXED POLICY $M$)

For a fixed watermark policy $M_{\text{fix}}$, the loss reduces to standard distillation:

$$\mathcal{L}_{\text{fix}}(\theta) = \text{KL}\left(\pi_o + \delta M_{\text{fix}} \parallel \pi_\theta\right). \tag{7}$$

Under Lipschitz continuity and bounded gradients, gradient descent on $\theta$ converges:

$$\lim_{t\to\infty} \|\nabla_\theta \mathcal{L}_{\text{fix}}\| = 0. \tag{8}$$

This serves as the baseline we extend.

### D.1.2 CAWP'S JOINT OPTIMIZATION

CAWP introduces a trainable policy $M_\phi$, leading to the joint loss:

$$\mathcal{L}_{\text{sim}}(\theta, \phi) = \underbrace{\text{KL}\left(\pi_o + \delta M_\phi \parallel \pi_\theta\right)}_{\text{Distillation loss}} + \lambda_2 \underbrace{\mathcal{L}_{\text{norm}}(\phi)}_{\text{Regularizer}}. \tag{9}$$

We analyze the effect of $M_\phi$ on convergence. Specifically, $\mathcal{L}_{\text{sim}}(\theta, \phi)$ satisfies the following assumptions:

- **Smoothness:** $\nabla\text{KL}(\cdot \parallel \pi_\theta)$ is $L_\theta$-Lipschitz in $\theta$, and $\nabla M_\phi$ is $L_\phi$-Lipschitz in $\phi$. LLMs and MLPs with smooth activations (e.g., GELU, Tanh) satisfy this.
- **Boundedness:** $\|M_\phi\| \leq B_M$ and $\|\nabla_\phi M_\phi\| \leq G_M$, since outputs are bounded by the last Tanh layer and gradients are bounded via clipping.

- **Convexity:** $\mathcal{L}_{\text{norm}}$ is convex in $\phi$, as $\ell_1$ penalties in Eq. (4) enforce convexity.

Given these assumptions, alternating gradient descent on $\theta$ and $\phi$ converges to a stationary point, where $M_\phi$ learns watermark mappings that perturb $\pi_o$ without degrading the LLM's performance.

#### D.1.3 Alternating Gradient Descent Dynamics

In CAWP, optimization alternates between updating $\theta$ (distilling to the perturbed teacher) and $\phi$ (adapting the watermark policy):

$$\theta^{t+1} = \theta^t - \eta_\theta \nabla_\theta \mathcal{L}_{\text{sim}}(\theta^t, \phi^t), \tag{10}$$

$$\phi^{t+1} = \phi^t - \eta_\phi \nabla_\phi \mathcal{L}_{\text{sim}}(\theta^{t+1}, \phi^t). \tag{11}$$

The distillation term encourages $\pi_\theta$ to match the perturbed distribution, while $\mathcal{L}_{\text{norm}}(\phi)$ (e.g., $\ell_1$ on policy outputs) prevents $M_\phi$ from over-perturbing, ensuring watermark detectability without degrading text quality.

Under the smoothness assumption, the loss is $L$-smooth overall ($L = L_\theta + \delta L_\phi B_M$). Hence,

$$\begin{aligned}\mathcal{L}_{\text{sim}}(\theta^{t+1}, \phi^{t+1}) \leq \mathcal{L}_{\text{sim}}(\theta^t, \phi^t) - \left( \frac{\eta_\theta}{2} \|\nabla_\theta \mathcal{L}_{\text{sim}}\|^2 + \frac{\eta_\phi}{2} \|\nabla_\phi \mathcal{L}_{\text{sim}}\|^2 \right) \\ + \frac{L\eta_\theta^2}{2} \|\nabla_\theta \mathcal{L}_{\text{sim}}\|^2 + \frac{L\eta_\phi^2}{2} \|\nabla_\phi \mathcal{L}_{\text{sim}}\|^2.\end{aligned} \tag{12}$$

Choosing $\eta_\theta, \eta_\phi \leq 1/L$ ensures monotonic decrease:

$$\mathcal{L}_{\text{sim}}(\theta^{t+1}, \phi^{t+1}) \leq \mathcal{L}_{\text{sim}}(\theta^t, \phi^t) - c\Big( \|\nabla_\theta \mathcal{L}_{\text{sim}}\|^2 + \|\nabla_\phi \mathcal{L}_{\text{sim}}\|^2 \Big), \quad c > 0. \tag{13}$$

#### D.1.4 Convergence to Stationary Point

Summing the descent inequality over $T$ iterations gives:

$$\sum_{t=0}^{T-1} \left( \|\nabla_\theta \mathcal{L}_{\text{sim}}(\theta^t, \phi^t)\|^2 + \|\nabla_\phi \mathcal{L}_{\text{sim}}(\theta^t, \phi^t)\|^2 \right) \leq \frac{\mathcal{L}_{\text{sim}}(\theta^0, \phi^0) - \mathcal{L}_{\text{sim}}^*}{cT}, \tag{14}$$

where $\mathcal{L}_{\text{sim}}^*$ is the infimum of the loss. As $T \to \infty$, the gradients vanish:

$$\lim_{t\to\infty} \|\nabla_\theta \mathcal{L}_{\text{sim}}\| = 0, \quad \lim_{t\to\infty} \|\nabla_\phi \mathcal{L}_{\text{sim}}\| = 0. \tag{15}$$

The boundedness of $M_\phi$ and its gradients ensures that the perturbation $\delta M_\phi$ remains controlled, preventing divergence. Moreover, the $\mu$-strong convexity of $\mathcal{L}_{\text{norm}}$ implies that, for fixed $\theta$, the subproblem in $\phi$ is strongly convex, guaranteeing convergence to a unique minimizer $\phi^*(\theta)$ that balances watermark strength and regularization.

### D.2 Computational Overhead of FPL

In this section, we provide additional details on the computational overhead of the proposed Forgotten Perturbation-aware Learning (FPL). Although the optimization objective in Equation 6 involves second-order derivatives, we show that it can be solved efficiently without computing exact Hessian information.

$$\arg\min_{\pi_\theta} \mathcal{L}_{\text{sim}}(\pi_\theta) + \beta \left( \mathcal{L}_{\text{anti}}(\pi_\theta) - \mathcal{L}_{\text{anti}}\left( \pi_\theta - \alpha \frac{\nabla \mathcal{L}_{\text{anti}}(\pi_\theta)}{\|\nabla \mathcal{L}_{\text{anti}}(\pi_\theta)\|} \right) \right) \tag{16}$$

where $\mathcal{L}_{\text{anti}}(\cdot)$ denotes the forgetting loss on anti-watermarked texts. Intuitively, the second term measures the decrease of $\mathcal{L}_{\text{anti}}$ after one normalized fine-tuning step along the forgetting gradient. By minimizing this gap, the model not only learns a strong watermark but also remains robust to potential forgetting induced by user-side modifications.

To solve this perturbation minimization problem, we consider an iterative gradient method (e.g., SGD). By the chain rule, the update rule is:

$$\pi_\theta^{t+1} = \pi_\theta^t - \eta\Bigg(\nabla\mathcal{L}_{\text{sim}}(\pi_\theta^t) + \beta\Big(\nabla\mathcal{L}_{\text{anti}}(\pi_\theta^t) - \nabla\mathcal{L}_{\text{anti}}\Big(\pi_\theta^t - \alpha\frac{\nabla\mathcal{L}_{\text{anti}}(\pi_\theta^t)}{\|\nabla\mathcal{L}_{\text{anti}}(\pi_\theta^t)\|}\Big)\underbrace{\nabla\Big(\pi_\theta^t - \alpha\frac{\nabla\mathcal{L}_{\text{anti}}(\pi_\theta^t)}{\|\nabla\mathcal{L}_{\text{anti}}(\pi_\theta^t)\|}\Big)}_{\text{second-order term}}\Big)\Bigg) \tag{17}$$

where $\eta$ is the learning rate. The last factor involves a second-order term (i.e., Hessian information), which is expensive to compute. Following prior work (Finn et al., 2017; Rajeswaran et al., 2019), we approximate this second-order term as a constant. The update rule then simplifies to:

$$\pi_\theta^{t+1} = \pi_\theta^t - \eta\Big(\nabla\mathcal{L}_{\text{sim}}(\pi_\theta^t) + \beta\big(\nabla\mathcal{L}_{\text{anti}}(\pi_\theta^t) - \nabla\mathcal{L}_{\text{anti}}\Big(\pi_\theta^t - \alpha\frac{\nabla\mathcal{L}_{\text{anti}}(\pi_\theta^t)}{\|\nabla\mathcal{L}_{\text{anti}}(\pi_\theta^t)\|}\Big)\big)\Big) \tag{18}$$

With this approximation, FPL requires only three forward/backward passes per optimization step:

1. A forward pass to compute $\mathcal{L}_{\text{sim}}(\pi_\theta)$ on watermarked text.
2. A forward and backward pass to compute $\nabla\mathcal{L}_{\text{anti}}(\pi_\theta)$ on anti-watermarked text.
3. A forward pass to evaluate $\mathcal{L}_{\text{anti}}(\pi_\theta - \alpha\hat{g})$, where $\hat{g}$ is the normalized forgetting gradient.

Thus, the overhead introduced by FPL is minimal. Moreover, CAWP accelerates convergence during training, making the overall computational cost comparable to baseline methods.

## E Computational Efficiency Analysis

To address concerns about computational efficiency, we provide a detailed comparison with prior watermarking methods. Our proposed techniques do not incur significant overhead and, in fact, accelerate convergence.

**CAWP.** The first component, CAWP, introduces only a lightweight trainable MLP as the watermark mapping model $M$, which contains $1.16$M parameters—just $0.0388\%$ of a 3B-parameter LLM. This addition is negligible compared to the base LLM and does not substantially increase computational cost.

**FPL.** The second component, FPL, involves two additional forward and backward passes per iteration. However, by using a learning-friendly watermark, our method reduces the number of training steps needed to achieve high performance. For example, PRO reaches an AUC of $0.997$ within $2000$ steps, while KGW requires $5000$ steps to reach a maximum AUC of $0.991$.

**Wall-clock Training Time.** In practice, the total wall-clock training time remains comparable across methods. For 8B-parameter models trained on $4\times$A100 80GB GPUs, all methods (KGW, KTH, PRO) complete training in about 6 hours. Thus, despite the slightly higher per-iteration cost, PRO achieves better performance with no additional end-to-end training time.

## F Watermarking for Closed-source LLMs

### F.1 KGW

Formally, the KGW(Kirchenbauer et al., 2023a) decoding-based watermarking strategy is defined as:

$$f_w^{\text{KGW}}(p, x, \xi; k, \gamma, \delta) = \text{softmax}\left(\log(p) + \delta \cdot f_{\text{hash}}^{\text{KGW}}(x_{\text{len}(x)-k+1}, \ldots, x_{\text{len}(x)}; \xi, \gamma, |\mathcal{V}|)\right) \tag{19}$$

Here, $f_{\text{hash}}^{\text{KGW}}$ is a pseudorandom hash function parameterized by the key $\xi$, which hashes the previous $k$ tokens in the sequence and returns $g \in \{0,1\}^{|\mathcal{V}|}$, containing $\gamma \cdot |\mathcal{V}|$ ones and $(1-\gamma) \cdot |\mathcal{V}|$ zeros, encoding the green list. For $k > 1$, we use the Additive-LeftHash scheme to hash multiple tokens by summing their token IDs. When $k = 0$, $f_{\text{hash}}^{\text{KGW}}$ returns a fixed green list $g$, independent of the previous tokens(Zhao et al., 2023).

The KGW watermark detection function is given by:

$$f_d^{\text{KGW}}(x, \xi; \gamma) = 1 - F_B \left( \sum_{t=k+1}^{\text{len}(x)} f_{\text{hash}}^{\text{KGW}}(x_{t-k}, \ldots, x_{t-1}; \xi, \gamma, |\mathcal{V}|)_{x_t} \right) \tag{20}$$

where $F_B$ is the cumulative distribution function (CDF) of the binomial random variable $B \sim \text{Bin}(\text{len}(x) - k, \gamma)$. This is because the number of green list tokens in non-watermarked text follows this distribution.

### F.2 KTH

Formally, the KTH (Kuditipudi et al., 2023) decoding-based watermarking strategy is defined as:

$$f_w^{\text{KTH}}(p, x, \xi) = \text{onehot}\left( \arg\max_i \left( \frac{\xi_i^{(\text{len}(x))}}{p_i} \right), |\mathcal{V}| \right), \tag{21}$$

where $\xi = (\xi^{(1)}, \ldots, \xi^{(m)})$ is the key consisting of $m$ vectors, each $\xi^{(j)} \in [0,1]^{|\mathcal{V}|}$ with entries sampled uniformly. The one-hot vector deterministically selects the token that maximizes the ratio between the key vector and the model distribution.

To introduce diversity across generations from the same prompt, the key $\xi$ is randomly shifted by an offset $\tau$ before generation. This results in a shifted key $\xi' = (\xi^{(1+\tau \bmod m)}, \ldots, \xi^{(m+\tau \bmod m)})$ used in $f_w^{\text{KTH}}$. To systematically explore variability, a hyperparameter $s \in [1, m]$ is defined, representing the number of distinct shifts. The shift values are evenly spaced in $[1, m]$, forming the set $\tau = \{i \cdot \lfloor m/s \rfloor \mid 0 \le i < s\}$. A larger $s$ expands the space of possible outputs and increases generation diversity.

For watermark detection, we evaluate how well the candidate text $x$ aligns with the key $\xi$ using the following test statistic:

$$d(x, \xi) = \sum_{t=1}^{\text{len}(x)} \log(1 - \xi_{x_t}^{(t)}), \tag{22}$$

which measures the cumulative alignment cost between the text and the key. Lower values of $d(x, \xi)$ indicate stronger watermark evidence. To determine significance, the observed score is compared against a reference distribution built from non-watermarked text, following the fixed-reference procedure in. With a reference set of size $T$, the $p$-value is lower bounded by $\frac{1}{T+1}$.

## G Model Modification

Open-source LLM are subject to modifications for task adaptation or deployment efficiency. The most prevalent types of such modifications include quantization, pruning, merging, and fine-tuning.

### G.1 Quantization

Model quantization techniques have emerged as an essential approach for deploying LLMs on memory-constrained hardware. The central idea is to reduce the precision of model weights and activations, typically from 16-bit or 32-bit floating-point representations to lower-precision formats such as 8-bit or 4-bit integers, thereby decreasing both storage and computational costs.

Quantization methods can generally be divided into two categories: *zero-shot* and *optimization-based*. **Zero-shot** methods apply fixed quantization mappings without model-specific calibration, as seen in approaches like LLM.INT8() (Dettmers et al., 2022) and NF4 (Dettmers et al., 2023). **Optimization-based** methods aim to minimize the error introduced by quantization, typically by optimizing over

a calibration dataset. HQQ focuses on minimizing reconstruction error over model weights alone. More sophisticated methods such as GPTQ (Frantar et al., 2022) and AWQ (Lin et al., 2024) further minimize activation-level reconstruction errors, achieving higher fidelity in the quantized model outputs.

### G.2 PRUNING

Model pruning aims to reduce the memory and computational cost of LLMs by removing redundant parameters. Pruning methods can be categorized into **unstructured** and **structured** approaches. Unstructured pruning removes individual weights independently, allowing fine-grained sparsity patterns, while structured pruning removes entire groups of weights, such as rows, columns, attention heads, or layers, leading to more hardware-friendly sparsity. Pruning methods aim to minimize a reconstruction loss between the outputs of the original dense model $f(\cdot;\boldsymbol{\theta})$ and the pruned model $f(\cdot;\boldsymbol{\theta}')$ on a calibration dataset $\mathcal{D}$. This can be formulated as:

$$\min_{\boldsymbol{\theta}'} \sum_{x\in\mathcal{D}} \|f(x;\boldsymbol{\theta}) - f(x;\boldsymbol{\theta}')\|^2, \tag{23}$$

where $\boldsymbol{\theta}'$ denotes the pruned parameters, with many weights either set to zero or structurally removed.

Unstructured methods such as WANDA (Sun et al., 2023), SPARSEGPT (Frantar & Alistarh, 2023), and GBLM (Das et al., 2023) apply weight-level pruning by evaluating importance scores and removing weights individually. In contrast, structured pruning methods such as SHEARED LLAMA (Xia et al., 2023) and LLM-PRUNER (Ma et al., 2023) remove entire structural components in the model jointly (e.g., rows or columns of weight matrices), while still minimizing the reconstruction error.

### G.3 MODEL MERGING

Model merging techniques aim to construct a new model by combining the weights of multiple pretrained models. Prior work (Yang et al., 2024; Goddard et al., 2024) has demonstrated that such merging can effectively integrate knowledge from task-specific expert models into a single model, often preserving or enhancing performance across the combined tasks. A common approach is based on the concept of *task vectors*, which assumes that different capabilities in LLMs are encoded in orthogonal directions in parameter space, allowing them to be combined additively or through interpolation.

A particularly principled merging method is *Spherical Linear Interpolation* (SLERP), which interpolates two models along the shortest path on the hypersphere defined by their parameter vectors. Given two model weight vectors $\boldsymbol{\theta}_1$ and $\boldsymbol{\theta}_2$, the angle $\Omega$ between them, and an interpolation parameter $t \in [0, 1]$, SLERP is defined as:

$$\text{SLERP}(\boldsymbol{\theta}_1, \boldsymbol{\theta}_2, t) = \frac{\sin[(1-t)\Omega]}{\sin\Omega}\boldsymbol{\theta}_1 + \frac{\sin[t\Omega]}{\sin\Omega}\boldsymbol{\theta}_2. \tag{24}$$

This formulation ensures that the interpolation stays on the unit sphere (assuming normalized weights), which help preserve model properties and stability during merging. If $\Omega = 0$ (i.e., the vectors are identical), the interpolation reduces to linear interpolation.

### G.4 FINE-TUNING

Model finetuning (Zhang et al., 2023; 2024)is a widely adopted approach for adapting pretrained language models to specific domains or tasks by continuing training on a smaller, curated dataset. It is particularly effective when the pretraining data distribution does not fully capture domain-specific or task-specific knowledge. Finetuning includes *Supervised Finetuning (SFT)* or *Instruction Tuning*. SFT involves training the model on explicit input–output pairs from a target task, while instruction tuning further generalizes this by training the model to follow natural language instructions, often across diverse tasks. Instruction tuning enhances a model's generalization and alignment with human intent, and is a key step in building instruction-following models capable of open-ended interaction.

Table 3: Watermarking hyperparameter configurations for LLaMA-8B.

| Method | Left | Middle | Right |
|---|---|---|---|
| Gu et al. (KGW) | $n = 0, \delta = 1$ | $n = 1, \delta = 1$ | $n = 1, \delta = 2$ |
| Gu et al. (KTH) | $s = 4$ | $s = 2$ | $s = 1$ |
| Gloaguen et al. (KGW) | $n = 0, \delta = 1$ | $n = 1, \delta = 1$ | $n = 1, \delta = 2$ |
| **PRO** | $\delta = 0.3$ | $\delta = 0.5$ | $\delta = 1$ |

Table 4: Watermarking hyperparameter configurations for LLaMA-3.2-3B.

| Method | Left | Middle | Right |
|---|---|---|---|
| Gu et al. (KGW) | $n = 0, \delta = 1$ | $n = 1, \delta = 1$ | $n = 1, \delta = 2$ |
| Gu et al. (KTH) | $s = 4$ | $s = 2$ | $s = 1$ |
| Gloaguen et al. (KGW) | $n = 0, \delta = 1$ | $n = 1, \delta = 1$ | $n = 1, \delta = 2$ |
| **PRO** | $\delta = 0.5$ | $\delta = 1$ | $\delta = 1.5$ |

Given a pretrained model with parameters $\boldsymbol{\theta}$ and a dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$, both SFT and instruction tuning optimize the empirical loss:

$$\min_{\boldsymbol{\theta}'} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(f(x_i; \boldsymbol{\theta}'), y_i), \tag{25}$$

Finetuning can be applied to all parameters or in a parameter-efficient manner using techniques such as LoRA or adapter layers, which reduce compute and memory costs while maintaining strong performance.

# H Experiments Configuration.

## H.1 Baseline Setting

Each table lists the hyperparameter configurations for the four open-source watermarking methods. The columns Left, Middle, and Right indicate the parameter settings used for the leftmost, middle, and rightmost points on each method's curve in the plots.

## H.2 Training Configurations

All models are fine-tuned on subsets of OpenWebText with different watermark strategies using a batch size of 64 sequences, sequence length of 512 tokens, a maximum learning rate of 1e-5 with cosine decay, and a linear warmup over the first 10% of steps. We use the AdamW optimizer with $(\beta_1, \beta_2) = (0.9, 0.999)$ and no weight decay. For KTH watermark distillation, we follow the same setup except for using a batch size of 128 and a sequence length of 256 tokens to accommodate memory constraints. For the Gloaguen et al. (Gloaguen et al., 2025)-KGW baseline, we follow their configuration. Starting from a model distilled with KGW, we fine-tune it on OpenWebText for 2,500 steps with a batch size of 64, sequence length of 512 tokens, a learning rate of 1e-5, and the AdamW optimizer. A cosine learning rate schedule is applied with 500 warmup steps. To identify stable parameters, we compute contrastive task vectors based on parameter change before and after fine-tuning, and perform a second-stage distillation restricted to these stable weights. Each training run took approximately 6 hours on 4 NVIDIA A100 80GB GPUs.

## H.3 Model Modification Settings

To assess robustness, we apply four types of model modifications that simulate real-world user behavior: (1) quantization, including INT8 (Dettmers et al., 2022) and GPTQ (Frantar et al., 2022) at 8-bit precision, GPTQ(Frantar et al., 2022) and HQQ (Badri & Shaji, 2023) at 4-bit; (2) unstructured pruning using WANDA (Sun et al., 2023) and SparseGPT (Frantar & Alistarh, 2023) at 20% and 50% sparsity levels; (3) model merging via SLERP (Goddard et al., 2024), where the

Table 5: Watermarking hyperparameter configurations for Phi-2-2.7B.

| Method | Left | Middle | Right |
|---|---|---|---|
| Gu et al. (KGW) | $n = 0, \delta = 1$ | $n = 1, \delta = 1$ | $n = 1, \delta = 2$ |
| Gu et al. (KTH) | $s = 4$ | $s = 2$ | $s = 1$ |
| Gloaguen et al. (KGW) | $n = 0, \delta = 1$ | $n = 1, \delta = 1$ | $n = 1, \delta = 2$ |
| **PRO** | $\delta = 0.5$ | $\delta = 1$ | $\delta = 1.5$ |

watermarked model is interpolated with its non-watermarked base model using mixing ratios from 0.1 to 0.9 following (Gloaguen et al., 2025); and (4) full-parameter fine-tuning on the task-specific `OPENMATHINSTRUCT` dataset (Toshniwal et al., 2024) and `OPENCODEINSTRUCT` dataset (Ahmad et al., 2025), reflecting the common use case where LLM users fine-tune open-source models on downstream data to build domain-specific experts. All modifications are implemented following the original settings of their respective methods.

# I Additional Experiments

## I.1 Visualization of ROC Curves

In Figure 7, we visualize the ROC curves for the proposed **PRO** and the state-of-the-art method by Gu et al.. We select watermarked LLaMA3-8B models with the same level of perplexity (i.e., 4.7). The results indicate that **PRO** achieves better watermark detectability while maintaining the same level of generation quality.

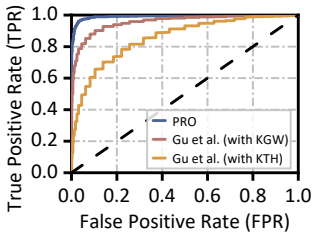


Figure 7: ROC curves for different methods.

Specifically, the ROC curve of **PRO** closely follows the top-left corner, indicating a higher true positive rate (TPR) across nearly all false positive rate (FPR) thresholds. In contrast, the methods by Gu et al. show noticeably lower TPRs, particularly at low FPR regions, which suggests a less reliable distinction between watermarked and non-watermarked texts under tight detection constraints. This improvement is especially evident in the early phase of the curve (e.g., FPR $< 0.1$), where **PRO** already achieves near-perfect detection performance, while the baseline methods lag behind. These findings further confirm that **PRO** not only embeds robust watermark patterns but also enables more confident detection, even at low error tolerance levels, making it more suitable for security-critical applications.

## I.2 More Dataset and Models

We run evaluations on an additional large language model, GPT-J-6B, using the same settings as in Figure 6 of the main paper. The results in Figure 8 show that the proposed **PRO** consistently outperforms other methods in terms of watermark detectability (i.e., AUC) and generation quality (i.e., Perplexity). Specifically, to achieve an AUC above 0.99, **PRO** maintains a perplexity of 17.8, while existing methods require at least a perplexity of 21.5.

## I.3 Detection Performance under Different FPRs

We further evaluate detection robustness by measuring the true positive rate (TPR) at different false positive rate (FPR) levels. This complements the AUC metric and highlights performance in stricter detection regimes. As shown in Table 6, **PRO** consistently outperforms prior methods, especially under very low FPRs (e.g., 0.1%), which are critical for practical deployment.

We run evaluations on an additional dataset of Wikipedia articles, using the same settings as in Figure 6 of the main paper, except for the dataset. We evaluate 5,000 completions of 200 tokens each, generated from 50-token prompts. As shown in Figure 9, the proposed **PRO** consistently achieves better watermark detectability at the same level of generation quality, indicating its generalizability to downstream users' prompts.

Table 6: Comparison of detection performance (TPR at different FPR levels) on LLaMA-3-8B. **PRO** demonstrates significantly stronger detection under stringent low-FPR conditions.

| Method | TPR@0.1% ↑ | TPR@1% ↑ | TPR@10% ↑ |
|---|---|---|---|
| Gu et al. (KTH) | 25.8% | 53.6% | 84.0% |
| Gu et al. (KGW) | 54.2% | 82.7% | 97.5% |
| **PRO** | **78.1%** | **92.3%** | **99.5%** |

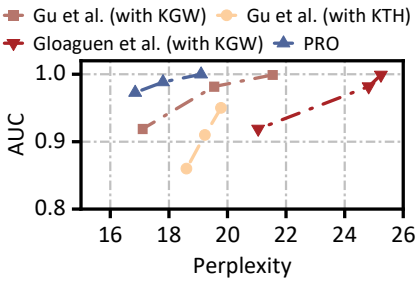


Figure 8: Results on GPT-J-6B.

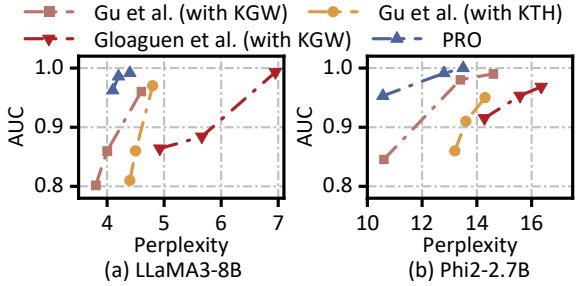


Figure 9: Results on Wikipedia dataset.

### I.4 Robustness against Paraphrasing Attack

We evaluate the robustness of our watermarking method under paraphrasing using DIPPER(Krishna et al., 2023), a controllable paraphraser that rewrites text while preserving semantics in Table 7. We use two settings: DIPPER-1, with lexical diversity set to 60 and order diversity 0; and DIPPER-2, with lexical diversity 60 and order diversity 20. Our method shows consistent detection performance under both settings, indicating robustness against paraphrase attacks.

Table 7: AUC under DIPPER attacks.

| Watermark Method | DIPPER1 | DIPPER2 |
|---|---|---|
| (Gu et al., 2023)-KTH | 0.82 | 0.79 |
| (Gu et al., 2023)-KGW | 0.86 | 0.84 |
| (Gloaguen et al., 2025)-KGW | 0.80 | 0.74 |
| **PRO (Ours)** | **0.90** | **0.87** |

### I.5 Robustness against Model Modifications across Downstream Tasks

We conducted experiments to evaluate FPL's effectiveness on more diverse downstream tasks. Specifically, we fine-tuned on the Alpaca dataset to further examine watermark robustness beyond code and math domains. As shown in Table 8, our method consistently outperforms the best baseline (Gloaguen et al. with KGW), with PRO's relative improvements shown in parentheses.

Table 8: Performance of watermarked LLaMA-3-8B under different fine-tuning steps $s$. The values in parentheses (prefixed with $+$) indicate the relative improvement compared to the baseline.

| Step $s$ | AUC | TPR@1% | TPR@10% |
|---|---|---|---|
| 300 | 0.91 (+0.02) | 54.3% (+2.8%) | 72.5% (+6.0%) |
| 600 | 0.87 (+0.04) | 49.4% (+4.0%) | 69.9% (+2.9%) |
| 900 | 0.85 (+0.02) | 40.5% (+9.1%) | 60.2% (+6.9%) |
| 1200 | 0.81 (+0.04) | 26.8% (+13.0%) | 44.7% (+13.6%) |
| 1500 | 0.79 (+0.05) | 16.9% (+16.5%) | 33.3% (+21.0%) |

### I.6 Compared with Classifier-based LLM Text Detectors

One distinct advantage of watermarking over classifier-based detectors is its ability to attribute text to a specific model, rather than just distinguishing LLM and human text. We evaluated PRO using the open-source classifier SuperAnnotate/ai-detector, which flagged 90.6% of PRO-generated texts as AI-generated. It also flagged 90% of outputs from non-watermarked and differently watermarked LLMs(Qwen-4B/8B, GPT-6B, LLaMA3-3B) as LLM-generated. To further analyze

this, we fine-tuned a binary RoBERTa classifier on texts from LLaMA3-8B and human texts. While it detected 60–80% of outputs from other LLMs (Qwen-4B/8B, GPT-6B, LLaMA3-3B) as LLaMA3-8B-generated. This shows that classifier-based methods primarily learn to distinguish LLM vs. human text, not between different LLMs In contrast, our PRO watermark enables model-level attribution. Specifically, we used the watermark policy model trained with LLaMA3-8B (student LLM) to detect outputs from other LLMs. The false positive rates remained low: 0.8% (Qwen-4B), 1.0% (Qwen-8B), 1.1% (GPT-6B), and 0.6% (LLaMA3-3B). This highlights PRO's precision in determining whether a text was generated by a specific model.

### I.7 Different Embedding Model and Watermark Mapping Model

To further validate the generality of our method, we supplemented the experiments with two additional embedding models: `thenlper/gte-large` and `intfloat/e5-large-v2`. Results in Table 9 show that our method maintains strong detection performance across different embeddings, consistently achieving near-perfect AUC and high TPR scores.

Table 9: Performance of PRO watermarking with LLaMA-3.2-3B under different embedding models.

| Embedding Model | AUC | TPR@1% | TPR@10% |
|---|---|---|---|
| `thenlper/gte-large` | 0.997 | 95.2% | 99.2% |
| `intfloat/e5-large-v2` | 0.994 | 94.7% | 99.0% |

We also compared different architectures for the watermark mapping model $M$. An MLP yields the best performance, as it can effectively exploit the full semantic embedding. By contrast, convolutional neural networks (CNNs) perform significantly worse, since semantic embeddings are single dense vectors without spatial or sequential structure for convolution to leverage. Thus, we adopt the MLP design in our framework.