MarkLLM: An Open-Source Toolkit for LLM Watermarking

Leyi Pan[1] Aiwei Liu[1] Zhiwei He[2] Zitian Gao[3] Xuandong Zhao[4] Yijian Lu[5]
Binglin Zhou[2] Shuliang Liu[6,7] Xuming Hu[6,7] Lijie Wen[1] Irwin King[5] Philip S. Yu[8]
[1]Tsinghua University [2]Shanghai Jiao Tong University [3]The University of Sydney
[4] UC Santa Barbara [5] The Chinese University of Hong Kong
[6] The Hong Kong University of Science and Technology (Guangzhou)
[7] The Hong Kong University of Science and Technology [8] University of Illinois at Chicago
panleyi2003@gmail.com, liuaw20@mails.tsinghua.edu.cn
xuminghu@hkust-gz.edu.cn, wenlj@tsinghua.edu.cn, king@cuhk.edu.hk, psyu@uic.edu

Abstract

Watermarking for Large Language Models (LLMs), which embeds imperceptible yet algorithmically detectable signals in model outputs to identify LLM-generated text, has become crucial in mitigating the potential misuse of LLMs. However, the abundance of LLM watermarking algorithms, their intricate mechanisms, and the complex evaluation procedures and perspectives pose challenges for researchers and the community to easily understand, implement and evaluate the latest advancements. To address these issues, we introduce MarkLLM, an open-source toolkit for LLM watermarking. MarkLLM offers a unified and extensible framework for implementing LLM watermarking algorithms, while providing user-friendly interfaces to ensure ease of access. Furthermore, it enhances understanding by supporting automatic visualization of the underlying mechanisms of these algorithms. For evaluation, MarkLLM offers a comprehensive suite of 12 tools spanning three perspectives, along with two types of automated evaluation pipelines. Through MarkLLM, we aim to support researchers while improving the comprehension and involvement of the general public in LLM watermarking technology, fostering consensus and driving further advancements in research and application. Our code is available at https://github.com/THU-BPM/MarkLLM.

1 Introduction

The emergence of Large Language Models (LLMs) like ChatGPT (OpenAI, 2022), GPT-4 (OpenAI, 2023), and LLaMA (Touvron et al., 2023) has significantly enhanced various tasks, including information retrieval (Zhu et al., 2023), content comprehension (Xiao et al., 2023), and creative writing (Gómez-Rodríguez and Williams, 2023). However, in the digital era, the remarkable proficiency of LLMs in generating high-quality text has also brought several issues to the forefront, including individuals impersonation (Salewski et al., 2023), academic paper ghostwriting (Vasilatos et al., 2023), and the proliferation of LLM-generated fake news (Megías et al., 2021). These issues highlight the urgent need for reliable methods to distinguish between human and LLM-generated content, particularly to prevent the spread of misinformation and ensure the authenticity of digital communication. In the light of this, LLM watermarking technology has been developed as a promising solution. By incorporating distinct features during the text generation process, LLM outputs can be uniquely identified using specially designed detectors.

As a developing technology, LLM watermarking urgently requires consensus and support from both within and outside the field. However, due to the proliferation of watermarking algorithms, their relatively complex mechanisms, the diversity of evaluation perspectives and metrics, as well as the intricate procedure of evaluation process, significant efforts are required by both researchers and the general public to easily experiment with, comprehend, and evaluate watermarking algorithms.

To bridge this gap, we introduce MarkLLM, an open-source toolkit for LLM watermarking. Figure 1 overviews the architecture of MarkLLM. Our main contributions are summarized as follows:

1) From a Functional Perspective:

- Implementation framework: MarkLLM offers a unified and extensible framework for implementing LLM watermarking algorithms, currently supporting nine specific algorithms from two key families: KGW (Kirchenbauer et al., 2023) and Christ (Christ et al., 2023) family.

- Unified top-calling interfaces: MarkLLM provides consistent, user-friendly interfaces for loading algorithms, producing watermarked text generated by LLMs, conducting detection processes, and gathering data necessary for visualization.

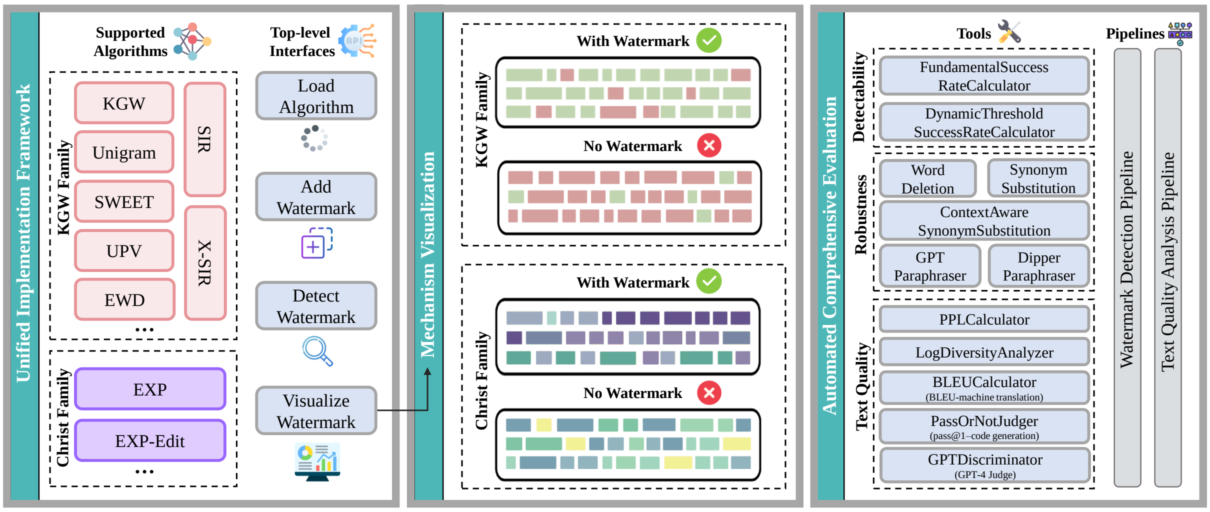

Figure 1: Architecture overview of MARKLLM.

- Visualization solutions: Custom visualization solutions are provided for both major watermarking algorithm families, enabling users to visualize the mechanisms of different algorithms under various configurations with real-world examples.

- Evaluation module: The toolkit includes 12 evaluation tools that address three critical perspectives: detectability, robustness, and impact on text quality. It also features two types of automated evaluation pipelines that support user customization of datasets, models, evaluation metrics and attacks, facilitating flexible and comprehensive assessments.

2) From a Design Perspective: MARKLLM is designed with a modular, loosely coupled architecture, ensuring its scalability and flexibility. This design choice facilitates the integration of new algorithms, the addition of innovative visualization techniques, and the extension of the evaluation toolkit by future developers.

3) From an Experimental Perspective: Utilizing MARKLLM as a research tool, we perform in-depth evaluations of the performances of the nine included algorithms, offering substantial insights and benchmarks that will be invaluable for ongoing and future research in LLM watermarking.

4) From an Ecosystem Perspective: MARKLLM provides a comprehensive set of resources, including an installable Python package (a GitHub repository and a pip package) with detailed installation and usage instructions, and an online Jupyter notebook demo hosted on Google Colab. Since its initial release, MARKLLM has garnered significant attention from researchers and developers, who have actively engaged with the project through stars, forks, issues, and pull requests, fostering continuous development and improvement. Figure 2 depicts the evolution of the MARKLLM ecosystem since its initial release. Due to the scope of this paper, we focus on presenting the core functionalities of MARKLLM, while acknowledging the broader ecosystem and community contributions that have emerged around the project.

2 Background

2.1 LLM Watermarking Algorithms

LLM watermarking methods can be broadly categorized into two major families: the KGW Family and the Christ Family. The KGW Family modifies the logits produced by the LLM to generate watermarked output, while the Christ Family alters the sampling process of LLM text generation to achieve watermarking.

The KGW method, as described by (Kirchenbauer et al., 2023), involves partitioning the vocabulary set into a green list and a red list based on the preceding token. During text generation, bias is added to the logits of green list tokens, leading to a preference for these tokens in the generated text. A statistical metric, based on the proportion of green words, is then calculated, and a corresponding threshold is set to differentiate watermarked from non-watermarked text. Building on this foundation, various modifications have been proposed to refine list partitioning or logit manipulation, aiming to improve the algorithm's performance in low-

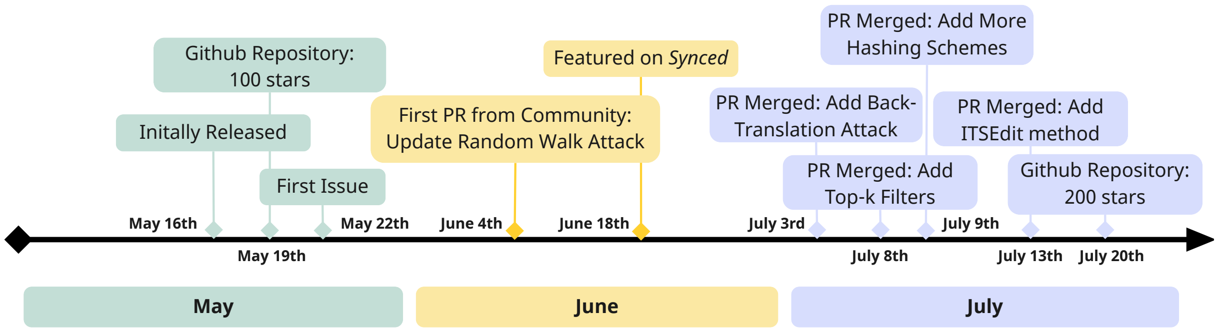

Figure 2: Timeline of the MarkLLM ecosystem since its initial release.

entropy settings (Lee et al., 2023; Lu et al., 2024), reduce the impact on text quality (Hu et al., 2024; Wu et al., 2023; Takezawa et al., 2023), increase the information capacity of the watermark (Wang et al., 2023; Yoo et al., 2023; Fernandez et al., 2023), counteract watermark removal attacks (Zhao et al., 2024; Liu et al., 2024b; Ren et al., 2023; He et al., 2024; Zhang et al., 2024), and enable public detection (Liu et al., 2024a; Fairoze et al., 2023).

Christ et al. (2023) introduced a method using a sequence of pseudo-random numbers to guide the sampling process in a binary LLM with a vocabulary of only 0s and 1s, resulting in detectable watermarks due to the correlation between the generated text and the sequence. On the other hand, Aaronson and Kirchner (2022) developed a watermarking algorithm suitable for real-world LLMs, which uses EXP-sampling. In this approach, a pseudo-random sequence of real numbers $r_1, ..., r_K \in [0, 1]$ is generated based on previous tokens: $r_i := f_s(w_{t-n+1}, ..., w_{t-1}, i)$, where $f_s()$ is a pseudo-random function. The token i is then selected to maximize r_i^{1/p_i} from the probability distribution $p_1, ..., p_K$ of the next token w_t. To detect a watermark, the sum $\sum_{t=1}^{T} \ln \frac{1}{1-r'_t}$ (where $r'_t = f_s(w_{t-n+1}, ..., w_t)$) measures the correlation between the text and the pseudo-random sequence, allowing for effective identification of watermarks by setting a suitable threshold. To further enhance robustness, Kuditipudi et al. (2023) suggested using edit distance to evaluate the correlation for detection.

2.2 Evaluation Perspectives

Evaluating the effectiveness of an algorithm entails considerations across various dimensions (Liu et al., 2023). Beyond the selection of different datasets and LLMs for text generation, three evaluation perspectives are crucial:

1) Watermark Detectability: This represents a fundamental property of an algorithm, indicating its capability to effectively discern watermarked LLM-generated text from natural content.

2) Robustness Against Tampering Attacks: An effective watermarking algorithm should embed watermarks in a manner that withstands minor modifications—like synonym substitution or paraphrasing—allowing the watermark to still be detectable by detectors with high reliability.

3) Impact on Text Quality: Watermarking algorithms intervene in LLM text generation processes and may affect the quality of the resulting text. This impact can be measured by metrics such as perplexity and output diversity, and by comparing the performance of the watermarked LLM against an unaltered LLM in specific downstream tasks.

3 MarkLLM

3.1 Unified Implementation Framework

So far, many watermarking algorithms have been proposed. However, as each algorithm implementation prioritizes its specific requirements over standardization, several issues have arisen:

1) Lack of Standardization in Class Design: This necessitates significant effort when optimizing or extending existing methods due to insufficiently standardized class designs.

2) Lack of Uniformity in Top-Level Calling Interfaces: The inconsistency in interfaces makes batch processing and replicating different algorithms cumbersome and labor-intensive.

3) Code Standard Issues: Challenges include the need to modify settings across multiple code segments and a lack of consistent documentation, which complicate the customization and effective use of the algorithms. Additionally, hard-coded values and inconsistent error handling can hinder adaptability and debugging efforts.

To address these issues, our toolkit offers a unified implementation framework that enables the convenient invocation of various state-of-the-art algorithms under flexible configurations. Additionally, our meticulously designed class structure paves the way for future extensions. Figure 3 demonstrates the design of the unified implementation framework.

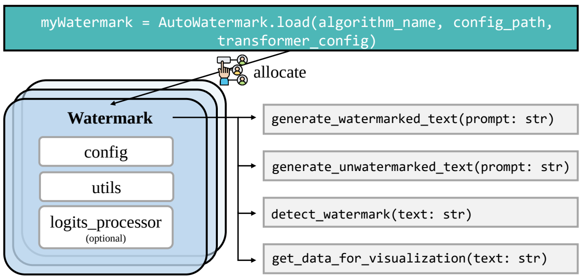

Figure 3: Unified implementation framework of LLM watermarking algorithms.

AutoWatermark.[1] This class is responsible for algorithm allocation. In its *.load()* method, it precisely locates the corresponding algorithm class using *algorithm_name* and accesses its configuration[2] for initialization via *config_path*. The method returns a fully configured algorithm object, thereby facilitating easy loading and efficient switching between different algorithms.

Watermark. Each watermarking algorithm has its own class, collectively referred to as the Watermark class. This class includes three data members: *config*, *utils*, and *logits_processor* (only for algorithms in the KGW Family). *config* holds algorithm parameters that are loaded from a configuration file, while *utils* comprises various helper functions and variables essential for algorithm operations. For algorithms within the KGW Family, *logits_processor* is specifically designed to manipulate logits and is integrated into *model.generate()* for processing during execution.

Top-level Interfaces. As illustrated in Figure 3, each algorithm has four top-level interfaces for generating watermarked text, generating unwatermarked text, detecting watermarks, and obtaining data for visualization (detailed in Section 3.2). Due to the framework's distributive design using an AutoWatermark class to allocate and return specific algorithm objects, developers can easily add interfaces to any algorithm class without impacting others. For an introduction to the algorithms integrated into the framework, please see Appendix A.

3.2 Mechanism Visualization

To improve understanding of the mechanisms used by different watermark algorithms, we have developed a visualization module that provides tailored visualization solutions for the two algorithm families.

3.2.1 Visualization Solutions

KGW Family. As detailed in Section 2.1, KGW family algorithms manipulate LLM output logits to prefer green tokens over red ones and employ statistical methods for detection. Our visualization technique clearly highlights red and green tokens in the text, offering insights into the token-level detection results.

Christ Family. Algorithms within Christ family involves guiding each token selection via a pseudo-random sequence and detect watermark by correlating the sequence with the textual content. To visualize this mechanism, we use a color gradient to express the correlation value, wherein darker shades signify stronger alignment. To quantify alignment for individual tokens, we utilize the formula $s = \ln \frac{1}{1-r'_t}$, as elaborated in Section 2.1. As the range of s spans from $[0, +\infty)$ while the color axis confines to $[0, 1]$, a monotonically increasing normalization function $m = \frac{s}{s+1}$ is applied to express alignment values. This transformation ensures that m remains within the range $[0, 1]$, while preserving the property that higher s values correspond to stronger alignment represented by higher m values.

3.2.2 Architecture Design

This section offers a detailed description of the architectural frameworks essential for the effective implementation of the aforementioned visualization strategies. Figure 4 demonstrates the implementation framework of mechanism visualization.

get_data_for_visualization: This interface, defined for each algorithm, returns a Visualization-

[1]There is a *transformers_config* parameter in the *.load()* method, which is an instance of the TransformersConfig class containing necessary information such as the model and tokenizer required for text generation. This parameter follows the naming conventions and specifications of the transformers library for model, tokenizer, and generate kwargs.

[2]For each watermarking algorithm, all user-modifiable parameters are consolidated into a dedicated configuration file, facilitating easy modifications.

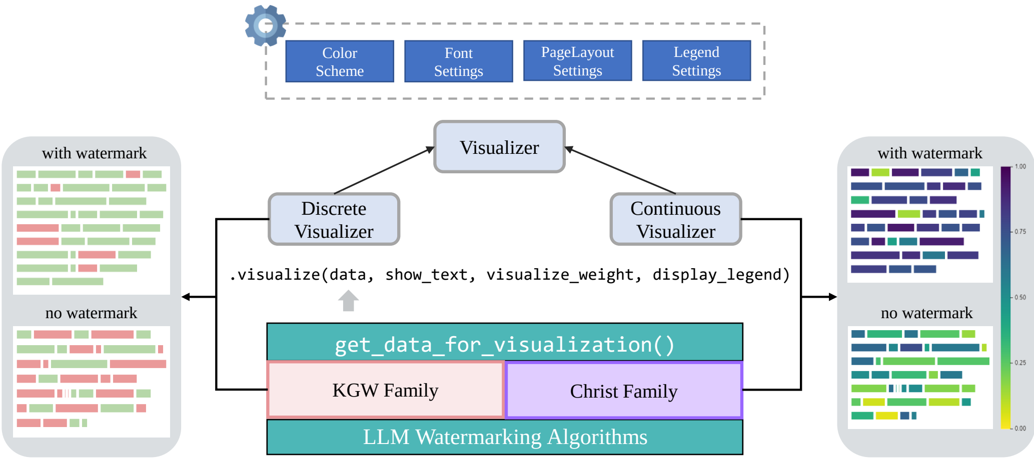

Figure 4: Implementation framework of mechanism visualization.

Data object containing *decoded_tokens* and *highlight_value*. For the KGW family, *highlight_value* is one-hot, differentiating red and green tokens; for the Christ family, it represents a continuous correlation value.

Visualizer: It initializes with a VisualizationData object and performs visualization via the *.visualize()* method, with subclasses overriding approach to implement specific visualizations.

DiscreetVisualizer: Tailored for KGW family algorithms, it uses red/green highlight values to color-code text based on values.

ContinuousVisualizer: Tailored for Christ family algorithms, it highlights tokens using a [0,1] color scale based on their alignment with pseudo-random numbers.

Flexible Visualization Settings: Our Visualizer supports multiple configurable options for tailored visualizations, including ColorScheme, FontSettings, PageLayoutSettings, and LegendSetting, allowing for extensive customization.

Minor: Our visualization design accommodates weighted differences among tokens during detection, as detailed in Appendix B.

3.2.3 Visualization Result

KGW Family. As illustrated in the leftmost part of Figure 4, in the text with watermarks, there is a relatively high proportion of green tokens. The statistical measure z-score is defined by the formula:

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

where $|s|_G$ denotes the number of green tokens, T represents the total number of tokens counted, and γ is a configuration setting representing the proportion of the green token list in partitioning, which in this case is 0.5. It is apparent that the z-score for 'text with watermark' is notably higher than that for 'text without watermark'. Therefore, setting a reasonable z-score threshold can effectively distinguish between the two.

Christ Family. As depicted in the rightmost part of Figure 4, it is noticeable that tokens within text containing watermarks generally exhibit darker hues compared to those without, indicating a higher influence of the sequence during the generation process on the former.

3.3 Automated Comprehensive Evaluation

Evaluating a LLM watermarking algorithm is a complex undertaking. Firstly, as mentioned in Section 2.2, evaluating an algorithm entails considering various perspectives, including watermark detectability, robustness against tampering, and impact on text quality. Secondly, evaluations from each perspective may necessitate different metrics, attack scenarios, and tasks. Additionally, conducting an evaluation typically entails multiple steps, such as model and dataset selection, watermarked text generation, post-processing, watermark detection, text tampering, and metric computation.

To facilitate convenient and thorough evaluation of LLM watermarking algorithms, MarkLLM offers twelve user-friendly tools, including various metric calculators and attackers that cover the three aforementioned evaluation perspectives. Additionally, MarkLLM provides two types of automated demo pipelines, whose modules can be customized and assembled flexibly, allowing for easy configuration and use.

Table 1: Evaluation Tools in MarkLLM.

Perspective	Tools
Detectability	FundamentalSuccessRateCalculator
	DynamicThresholdSuccessRateCalculator
Robustness	WordDeletion
	SynonymSubstitution
	ContextAwareSynonymSubstitution
	GPTParaphraser
	DipperParaphraser
Text Quality	PPLCaluclator
	LogDiversityAnalyzer
	BLEUCalculator
	PassOrNotJudger
	GPTDiscriminator

Evaluation Tools. Table 1 summarizes all the tools currently supported in MarkLLM.

For the aspect of detectability, most watermarking algorithms ultimately require specifying a threshold to distinguish between watermarked and non-watermarked texts. We provide a basic success rate calculator using a fixed threshold. Additionally, to minimize the impact of threshold selection on detectability, we also offer a calculator that supports dynamic threshold selection. This tool can determine the threshold that yields the best F1 score or select a threshold based on a user-specified target false positive rate (FPR).

For the aspect of robustness, MarkLLM offers three word-level text tampering attacks: random word deletion at a specified ratio, random synonym substitution using WordNet (Miller, 1995) as the synonym set, and context-aware synonym substitution utilizing BERT (Devlin et al., 2018) as the embedding model. Additionally, two document-level text tampering attacks are provided: paraphrasing the context via OpenAI API or the Dipper model (Krishna et al., 2023).

For the aspect of text quality, MarkLLM offers two direct analysis tools: a perplexity calculator to gauge fluency and a diversity calculator to evaluate the variability of texts. To analyze the impact of watermarking on text utility in specific downstream tasks, we provide a BLEU calculator for machine translation tasks and a pass-or-not judger for code generation tasks. Additionally, given the current methods for comparing the quality of watermarked and unwatermarked text, which include using a stronger LLM for judgment (Tu et al., 2023), we also offer a GPT discriminator, utilizing GPT-4 (OpenAI, 2023) to compare text quality.

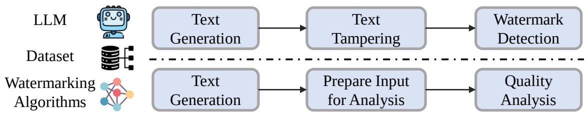

Figure 5: The standardized process of evaluation pipelines, the upper for watermark detection pipeline, and the lower for text quality analysis pipeline.

Evaluation Pipelines. To facilitate automated evaluation of LLM watermarking algorithms, MarkLLM provides two evaluation pipelines: one for assessing watermark detectability with and without attacks, and another for analyzing the impact of these algorithms on text quality.

The upper part of Figure 5 illustrates the standardized process of watermark detection. Following this process, we have implemented two pipelines: **WMDetect**[3] and **UWMDetect**[4]. The primary difference between them lies in the text generation phase. The former requires the use of the *generate_watermarked_text* method from the watermarking algorithm, while the latter depends on the text_source parameter to determine whether to directly retrieve natural text from a dataset or to invoke the *generate_unwatermarked_text* method.

The lower part of Figure 5 illustrates the unified process of text quality analysis. To evaluate the impact of watermarking on text quality, pairs of watermarked and unwatermarked texts are generated. The texts, along with other necessary inputs, are then processed and fed into a designated text quality analyzer to produce detailed analysis and comparison results. Following this process, we have implemented three pipelines for different evaluation scenarios.

DirectQual.[5] This pipeline is specifically designed to analyze the quality of texts by directly comparing the characteristics of watermarked texts with those of unwatermarked texts. It evaluates metrics such as perplexity (PPL) and log diversity, without the need for any external reference texts.

RefQual.[6] This pipeline evaluates text quality by comparing both watermarked and unwatermarked texts with a common reference text. It measures the degree of similarity or deviation from the reference text. It is ideal for scenarios that require

[3]Short for 'Watermarked Text Detection Pipeline'.
[4]Short for 'Unwatermarked Text Detection Pipeline'.
[5]Short for 'Direct Text Quality Analysis Pipeline'.
[6]Short for 'Referenced Text Quality Analysis Pipeline'.

specific downstream tasks to assess text quality, such as machine translation and code generation.

ExDisQual.[7] This pipeline employs an external judger, such as GPT-4 (OpenAI, 2023), to assess the quality of both watermarked and unwatermarked texts. The discriminator evaluates the texts based on user-provided task descriptions, identifying any potential degradation or preservation of quality due to watermarking. This method is particularly valuable when an advanced, AI-based analysis of the subtle effects of watermarking is required.

4 User Examples

The following code snippets demonstrate examples of how to use MarkLLM in one's project. For more real cases, please see the demo video.

4.1 Watermarking Algorithm Invocation

```
# Load algorithm
myWatermark = AutoWatermark.load('KGW', 'config/KGW.json', transformers_config)

# Generate watermarked text
watermarked_text = myWatermark.generate_watermarked_text(prompt)

# Detect watermark
detect_result = myWatermark.detect_watermark(watermarked_text)
```

4.2 Mechanism Visualization

```
# Get data for visualization
watermarked_data = myWatermark.get_data_for_visualization(watermarked_text)

# Init visualizer
visualizer = DiscreetVisualizer(ColorSchemeForDiscreetVisualization(), FontSettings(), PageLayoutSettings(), DiscreetLegendSettings())

# Visualize
watermarked_img = visualizer.visualize(watermarked_data)
```

4.3 Evaluation Pipelines Invocation

```
# Dataset
my_dataset = C4Dataset('dataset/c4/processed_c4.json')

# Watermarked text detection pipeline
pipeline1 = WatermarkedTextDetectionPipeline(my_dataset)

# Unwatermarked text detection pipeline
pipeline2 = UnWatermarkedTextDetectionPipeline(dataset=my_dataset)

# Init calculator
calculator = DynamicThresholdSuccessRateCalculator(labels=['TPR', 'F1'], rule='best')

# Calculate success rate
print(calculator.calculate(pipeline1.evaluate(my_watermark), pipeline2.evaluate(my_watermark)))
```

[7] Short for 'External Discriminator Text Quality Analysis Pipeline'.

5 Experiment

Using MARKLLM as a research tool, we conduct evaluations on nine algorithms, assessing their detectability, robustness, and impact on text quality. Our experiments aim to showcase MARKLLM's effectiveness and efficiency through practical case studies.

5.1 Experiment Settings

Dateset and Prompt. For general-purpose text generation scenarios, we utilize the C4 dataset (Raffel et al., 2020). Specifically, the first 30 tokens of texts serve as prompts for generating the subsequent 200 tokens, with the original C4 texts acting as non-watermarked examples. For specific downstream tasks, we employ the WMT16 (Bojar et al., 2016) German-English dataset for machine translation, and HumanEval (Chen et al., 2021) for code generation.

Language Model. For general-purpose text generation scenarios, we utilize Llama-7b (Touvron et al., 2023) as language model. For specific downstream tasks, we utilize NLLB-200-distilled-600M (Costa-jussà et al., 2022) for machine translation and Starcoder (Li et al., 2023) for code generation.

Metrics and Attacks. Dynamic threshold adjustment is employed to evaluate watermark detectability, with three settings provided: under a target FPR of 10%, under a target FPR of 1%, and under conditions for optimal F1 score performance. To assess robustness, we utilize all text tampering attacks listed in Table 1. For evaluating the impact on text quality, our metrics include PPL, log diversity, BLEU (for machine translation), pass@1 (for code

Table 2: The evaluation results of assessing the detectability of nine algorithms supported in MarkLLM. 200 watermarked texts are generated, while 200 non-watermarked texts serve as negative examples. We furnish TPR and F1-score under dynamic threshold adjustments for 10% and 1% FPR, alongside TPR, TNR, FPR, FNR, P, R, F1, ACC at optimal performance.

Method	10%FPR		1%FPR		Best							
	TPR	F1	TPR	F1	TPR	TNR	FPR	FNR	P	R	F1	ACC
KGW	**1.000**	0.952	**1.000**	0.995	**1.000**	**1.000**	**0.000**	**0.000**	**1.000**	**1.000**	**1.000**	**1.000**
Unigram	**1.000**	**0.957**	**1.000**	**0.995**	**1.000**	**1.000**	**0.000**	**0.000**	**1.000**	**1.000**	**1.000**	**1.000**
SWEET	**1.000**	0.952	**1.000**	**0.995**	**1.000**	**1.000**	**0.000**	**0.000**	**1.000**	**1.000**	**1.000**	**1.000**
UPV	×	×	×	×	**1.000**	0.990	0.010	**0.000**	0.990	**1.000**	0.995	0.995
EWD	**1.000**	0.952	**1.000**	**0.995**	0.995	**1.000**	**0.000**	0.005	**1.000**	0.995	0.997	0.998
SIR	0.995	0.950	0.990	0.990	0.990	0.995	0.005	0.010	0.995	0.990	0.992	0.993
X-SIR	0.995	0.950	0.940	0.964	0.970	0.970	0.030	0.030	0.970	0.970	0.970	0.970
EXP	**1.000**	0.952	**1.000**	**0.995**	**1.000**	**1.000**	**0.000**	**0.000**	**1.000**	**1.000**	**1.000**	**1.000**
EXP-Edit	**1.000**	0.952	0.995	0.990	0.995	0.985	0.015	0.005	0.985	0.995	0.990	0.990

Table 3: The evaluation results of assessing the robustness of nine algorithms supported in MarkLLM. For each attack, 200 watermarked texts are generated and subsequently tampered, with an additional 200 non-watermarked texts serving as negative examples. We report the TPR and F1-score at optimal performance under each circumstance.

Method	No Attack		Word-D		Word-S		Word-S (Context)		Doc-P (GPT-3.5)		Doc-P (Dipper)	
	TPR	F1	TPR	F1	TPR	F1	TPR	F1	TPR	F1	TPR	F1
KGW	**1.000**	**1.000**	0.980	0.985	0.920	0.915	0.965	0.958	0.835	0.803	0.860	0.785
Unigram	**1.000**	**1.000**	**1.000**	**1.000**	**0.990**	**0.990**	**0.990**	**0.990**	**0.901**	**0.932**	0.875	**0.908**
SWEET	**1.000**	**1.000**	0.970	0.975	0.935	0.903	0.985	0.980	0.845	0.813	0.830	0.779
UPV	**1.000**	0.995	0.970	0.980	0.885	0.896	0.985	0.961	0.830	0.827	0.862	0.864
EWD	0.995	0.997	0.980	0.982	0.930	0.921	0.950	0.955	0.852	0.825	0.845	0.784
SIR	0.990	0.992	0.950	0.970	0.945	0.940	0.960	0.948	0.891	0.923	**0.894**	0.902
X-SIR	0.970	0.970	0.940	0.957	0.910	0.908	0.895	0.925	0.875	0.891	0.835	0.869
EXP	**1.000**	**1.000**	0.975	0.980	0.945	0.950	0.980	0.985	0.763	0.772	0.740	0.793
EXP-Edit	0.995	0.990	0.995	0.993	0.983	0.972	0.990	0.985	0.872	0.886	0.845	0.861

Table 4: The evaluation results of assessing the text quality impact of the nine algorithms supported in MarkLLM. We compared 200 watermarked texts with 200 non-watermarked texts. However, due to dataset constraints, only 100 watermarked texts were compared with 100 non-watermarked texts for code generation.

Method	Direct Analysis		Referenced Analysis		External Discriminator
	PPL(Ori.= 8.243)	Log Diversity(Ori.=8.517)	Machine Translation BLEU(Ori.=31.807)	Code Generation pass@1(Ori.= 43.0)	Machine Translation GPT-4 Judge (Wat. Win Rate)
KGW	13.551 ↑	7.989 ↓	28.242 ↓	34.0 ↓	0.31
Unigram	13.723 ↑	7.242 ↓	26.075 ↓	32.0 ↓	**0.33**
SWEET	13.747 ↑	8.086 ↓	28.242 ↓	**37.0** ↓	0.31
UPV	**10.574** ↑	7.698 ↓	28.270 ↓	**37.0** ↓	0.31
EWD	13.402 ↑	8.220 ↓	28.242 ↓	34.0 ↓	0.30
SIR	13.918 ↑	7.990 ↓	**28.830** ↓	**37.0** ↓	0.31
X-SIR	12.885 ↑	7.930 ↓	28.161 ↓	36.0 ↓	**0.33**
EXP	19.597 ↑	8.187 ↓	×	20.0 ↓	×
EXP-Edit	21.591 ↑	**9.046** ↑	×	14.0 ↓	×

generation), and assessments using GPT-4 Judge (Tu et al., 2023).

Hyper-parameters. Configuration files for each algorithms are listed in Appendix D.1[8]. Parameter settings for the evaluation tools are listed in

[8]Note that each algorithm was tested using only one parameter configuration to demonstrate MARKLLM's functionality and provide preliminary reference data. Extensive performance comparisons across different aspects of each algorithm would require varied parameter settings and further experimentation.

Appendix D.2.

5.2 Results and Analysis

The results[9] in Table 2, Table 3, and Table 4 demonstrate that by using the implementations of different algorithms and the evaluation pipelines provided in MarkLLM, researchers can effectively reproduce the experimental results from previous watermarking papers. These experiments can be conducted by running simple scripts (detailed in Appendix C), showcasing MarkLLM's capability for easy evaluation of watermark algorithms in various scenarios. This highlights the tool's user-friendliness and practical utility, offering valuable insights for future research.

Through systematic evaluation, it can be observed that: (1) Current LLM watermarking algorithms excel in achieving accurate detection, boasting F1-scores surpassing 0.99 in no-attack conditions; (2) Different algorithms demonstrate distinct strengths across various aspects, necessitating consideration of specific circumstance when selecting an algorithm; (3) Even when evaluating from the same perspective, results can vary depending on the metrics or types of attacks used. This highlights the need for a thorough assessment when judging algorithms; (4) Striking a harmonious balance between various evaluation perspectives poses a significant challenge. Future research should prioritize balancing and enhancing the overall capabilities of algorithms.

In summary, MarkLLM acts as a convenient tool for conducting diverse evaluation experiments, effectively minimizing assessment expenses. Future research can leverage MarkLLM for comprehensive exploration and analysis.

6 Conclusion

MarkLLM is a comprehensive open-source toolkit for LLM watermarking. It allows users to easily try various state-of-the-art algorithms with flexible configurations to watermark their own text and conduct detection, and provides clear visualizations to gain insights into the underlying mechanisms. The inclusion of convenient evaluation tools and customizable evaluation pipelines enables automatic and thorough assessments from various perspectives. As LLM watermarking evolves, MarkLLM aims to be a collaborative platform that grows with the research community. By providing a solid foundation and inviting contributions, we aim to foster a vibrant ecosystem where researchers and developers can work together to advance the state-of-the-art in LLM watermarking technology.

Limitations

MarkLLM is a comprehensive toolkit for implementing, visualizing, and evaluating LLM watermarking algorithms. However, it currently only integrates a subset of existing watermarking methods and does not yet support some recent approaches that directly embed watermarks into model parameters during training (Xu et al., 2024; Gu et al., 2024). We anticipate future contributions to expand MarkLLM's coverage and enhance its versatility.

In terms of visualization, we have provided one tailored solution for each of the two main watermarking algorithm families. While these solutions offer valuable insights, there is room for more creative and diverse visualization designs.

Regarding evaluation, we have covered aspects such as detectability, robustness, and text quality impact. However, our current toolkit may not encompass all possible scenarios, such as retranslation and CWRA (He et al., 2024) attacks related to robustness.

We acknowledge that MarkLLM has room for improvement. We warmly welcome developers and researchers to contribute their code and insights to help build a more comprehensive and robust ecosystem for LLM watermarking. Through collaborative efforts, we can further advance this technology and unlock its full potential.

[9](1) The evaluation results for UPV are only shown in the "best" column because its watermark detection uses direct binary classification without thresholds. (2) Current implementations of Christ family algorithms are designed for decoder-only LLMs. As machine translation mainly uses encoder-decoder models, we did not report the text quality produced by EXP and EXP-edit in machine translation.

References

S. Aaronson and H. Kirchner. 2022. Watermarking gpt outputs. https://www.scottaaronson.com/talks/watermark.ppt.

Ond rej Bojar, Rajen Chatterjee, Christian Federmann, Yvette Graham, Barry Haddow, Matthias Huck, Antonio Jimeno Yepes, Philipp Koehn, Varvara Logacheva, Christof Monz, Matteo Negri, Aurelie Neveol, Mariana Neves, Martin Popel, Matt Post, Raphael Rubino, Carolina Scarton, Lucia Specia, Marco Turchi, Karin Verspoor, and Marcos Zampieri. 2016. Findings of the 2016 conference on machine translation. In *Proceedings of the First Conference on Machine Translation*, pages 131–198, Berlin, Germany. Association for Computational Linguistics.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code.

Miranda Christ, Sam Gunn, and Or Zamir. 2023. Undetectable watermarks for language models. *arXiv preprint arXiv:2306.09194*.

Marta R Costa-jussà, James Cross, Onur Çelebi, Maha Elbayad, Kenneth Heafield, Kevin Heffernan, Elahe Kalbassi, Janice Lam, Daniel Licht, Jean Maillard, et al. 2022. No language left behind: Scaling human-centered machine translation. *arXiv preprint arXiv:2207.04672*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Jaiden Fairoze, Sanjam Garg, Somesh Jha, Saeed Mahloujifar, Mohammad Mahmoody, and Mingyuan Wang. 2023. Publicly detectable watermarking for language models. Cryptology ePrint Archive, Paper 2023/1661. https://eprint.iacr.org/2023/1661.

Pierre Fernandez, Antoine Chaffin, Karim Tit, Vivien Chappelier, and Teddy Furon. 2023. Three bricks to consolidate watermarks for large language models. *arXiv preprint arXiv:2308.00113*.

Carlos Gómez-Rodríguez and Paul Williams. 2023. A confederacy of models: A comprehensive evaluation of llms on creative writing. *arXiv preprint arXiv:2310.08433*.

Chenchen Gu, Xiang Lisa Li, Percy Liang, and Tatsunori Hashimoto. 2024. On the learnability of watermarks for language models. In *The Twelfth International Conference on Learning Representations*.

Zhiwei He, Binglin Zhou, Hongkun Hao, Aiwei Liu, Xing Wang, Zhaopeng Tu, Zhuosheng Zhang, and Rui Wang. 2024. Can watermarks survive translation? on the cross-lingual consistency of text watermark for large language models. *arXiv preprint arXiv:2402.14007*.

Zhengmian Hu, Lichang Chen, Xidong Wu, Yihan Wu, Hongyang Zhang, and Heng Huang. 2024. Unbiased watermark for large language models. In *The Twelfth International Conference on Learning Representations*.

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, ICML 2023, 23-29 July 2023, Honolulu, Hawaii, USA*, volume 202 of *Proceedings of Machine Learning Research*, pages 17061–17084. PMLR.

John Kirchenbauer, Jonas Geiping, Yuxin Wen, Manli Shu, Khalid Saifullah, Kezhi Kong, Kasun Fernando, Aniruddha Saha, Micah Goldblum, and Tom Goldstein. 2024. On the reliability of watermarks for large language models. In *The Twelfth International Conference on Learning Representations*.

Kalpesh Krishna, Yixiao Song, Marzena Karpinska, John Wieting, and Mohit Iyyer. 2023. Paraphrasing evades detectors of ai-generated text, but retrieval is an effective defense. *arXiv preprint arXiv:2303.13408*.

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*.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2023. Starcoder: may the source be with you!

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. 2024b. A semantic invariant robust watermark for large language models. In *The Twelfth International Conference on Learning Representations*.

Aiwei Liu, Leyi Pan, Yijian Lu, Jingjing Li, Xuming Hu, Lijie Wen, Irwin King, and Philip S Yu. 2023. A survey of text watermarking in the era of large language models. *arXiv preprint arXiv:2312.07913*.

Yijian Lu, Aiwei Liu, Dianzhi Yu, Jingjing Li, and Irwin King. 2024. An entropy-based text watermarking detection method. *arXiv preprint arXiv:2403.13485*.

David Megías, Minoru Kuribayashi, Andrea Rosales, and Wojciech Mazurczyk. 2021. Dissimilar: Towards fake news detection using information hiding. In *Signal Processing and Machine Learning. In The 16th International Conference on Availability, Reliability and Security (Vienna, Austria)(ARES 2021). Association for Computing Machinery, New York, NY, USA, Article*, volume 66.

George A Miller. 1995. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41.

OpenAI. 2022. Chatgpt: Optimizing language models for dialogue. https://openai.com/blog/chatgpt.

OpenAI. 2023. Gpt-4 technical report. *ArXiv*, abs/2303.08774.

Julien Piet, Chawin Sitawarin, Vivian Fang, Norman Mu, and David Wagner. 2023. Mark my words: Analyzing and evaluating language model watermarks.

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. *The Journal of Machine Learning Research*, 21(1):5485–5551.

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

Leonard Salewski, Stephan Alaniz, Isabel Rio-Torto, Eric Schulz, and Zeynep Akata. 2023. In-context impersonation reveals large language models' strengths and biases.

Yuki Takezawa, Ryoma Sato, Han Bao, Kenta Niwa, and Makoto Yamada. 2023. Necessary and sufficient watermark for large language models. *arXiv preprint arXiv:2310.00833*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Shangqing Tu, Yuliang Sun, Yushi Bai, Jifan Yu, Lei Hou, and Juanzi Li. 2023. Waterbench: Towards holistic evaluation of watermarks for large language models. *arXiv preprint arXiv:2311.07138*.

Christoforos Vasilatos, Manaar Alam, Talal Rahwan, Yasir Zaki, and Michail Maniatakos. 2023. Howkgpt: Investigating the detection of chatgpt-generated university student homework through context-aware perplexity analysis. *arXiv preprint arXiv:2305.18226*.

Lean Wang, Wenkai Yang, Deli Chen, Hao Zhou, Yankai Lin, Fandong Meng, Jie Zhou, and Xu Sun. 2023. Towards codable text watermarking for large language models. *arXiv preprint arXiv:2307.15992*.

Yihan Wu, Zhengmian Hu, Hongyang Zhang, and Heng Huang. 2023. Dipmark: A stealthy, efficient and resilient watermark for large language models. *arXiv preprint arXiv:2310.07710*.

Changrong Xiao, Sean Xin Xu, Kunpeng Zhang, Yufang Wang, and Lei Xia. 2023. Evaluating reading comprehension exercises generated by llms: A showcase of chatgpt in education applications. In *Proceedings of the 18th Workshop on Innovative Use of NLP for Building Educational Applications (BEA 2023)*, pages 610–625.

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

KiYoon Yoo, Wonhyuk Ahn, and Nojun Kwak. 2023. Advancing beyond identification: Multi-bit watermark for language models. *arXiv preprint arXiv:2308.00221*.

Hanlin Zhang, Benjamin L. Edelman, Danilo Francati, Daniele Venturi, Giuseppe Ateniese, and Boaz Barak. 2024. Watermarks in the sand: Impossibility of strong watermarking for language models. In *Forty-first International Conference on Machine Learning*.

Xuandong Zhao, Prabhanjan Vijendra Ananth, Lei Li, and Yu-Xiang Wang. 2024. Provable robust watermarking for AI-generated text. In *The Twelfth International Conference on Learning Representations*.

Yutao Zhu, Huaying Yuan, Shuting Wang, Jiongnan Liu, Wenhan Liu, Chenlong Deng, Zhicheng Dou, and Ji-Rong Wen. 2023. Large language models for information retrieval: A survey. *arXiv preprint arXiv:2308.07107*.

A Supported LLM Watermarking Algorithms

Table 5 showcases the nine watermarking algorithms that have been integrated into MarkLLM. These algorithms were carefully selected to include representatives from both the KGW family and the Christ family, ensuring broad coverage across diverse optimization objectives such as enhancing detectability, improving robustness, bettering text quality, and enabling public detection.

Table 5: Details of watermarking algorithms supported in MarkLLM.

Algorithm Name	Category	Methodology
KGW (Kirchenbauer et al., 2023)	KGW Family	Separate the vocabulary set into two lists: a red list and a green list based on the preceding token, then add bias to green tokens so that the LLM-produced text exhibits preference of using green tokens.
Unigram (Zhao et al., 2024)	KGW Family	Use a globally fixed red-green list separation for every location, aiming to enhance robustness against tampering.
SWEET (Lee et al., 2023)	KGW Family	Modifications to the logits are selectively applied only to high-entropy locations while bypassing low-entropy ones, aiming to improve text quality in low-entropy scenarios, such as code generation.
UPV (Liu et al., 2024a)	KGW Family	Train one network as a generator for segregating red-green lists and another network as a detector for directly providing a classification result based on textual input. The distinction between the generator network and the detector network enables public detection.
EWD (Lu et al., 2024)	KGW Family	During the watermark detection phase, each token is assigned a different weight based on its entropy, with tokens having higher entropy receiving greater weight. This approach aims to enhance watermark detectability in low-entropy situations.
SIR (Liu et al., 2024b)	KGW Family	Train a generator network to convert token embeddings into context-aware biases, thereby enhancing robustness against semantic invariant tampering.
X-SIR (He et al., 2024)	KGW Family	The red-green partition of the vocabulary no longer operates at the token level but rather at the level of semantic clusters, grouping semantically similar words together within the same group and adding bias at the group level. This shift from adding bias to individual green tokens to green clusters is designed to enhance robustness against Cross-lingual Watermark Removal Attacks (CWRA).
EXP (Aaronson and Kirchner, 2022)	Christ Family	Utilize a pseudo-random sequence based on preceding tokens to guide token sampling when generating each new token through the exponential (EXP) sampling rule. As a result, the watermarked text displays a degree of alignment with the sequence, aiding in its detection.
EXP-Edit (Kuditipudi et al., 2023)	Christ Family	Expanding on the use of the EXP sampling rule, introduce the concept of edit distance to measure the alignment between the pseudo-random sequence and the text, which significantly improves its robustness against tampering.

B Handling Weighted Token Differences in Visualization

Currently, several LLM Watermarking algorithms assign different weights to each token during detection to adapt to more stringent usage environments. Consequently, we have also implemented the visualization of these weights. For instance, both the SWEET (Lee et al., 2023) and EWD (Lu et al., 2024) methods were developed to function effectively in low-entropy environments.

The SWEET method calculates the entropy of each token before watermarking it. If the entropy is below a certain threshold, the token is not watermarked, thus mitigating the problem of text quality degradation due to modifications in low entropy areas.Therefore, the weights assigned to each token are either 0 or

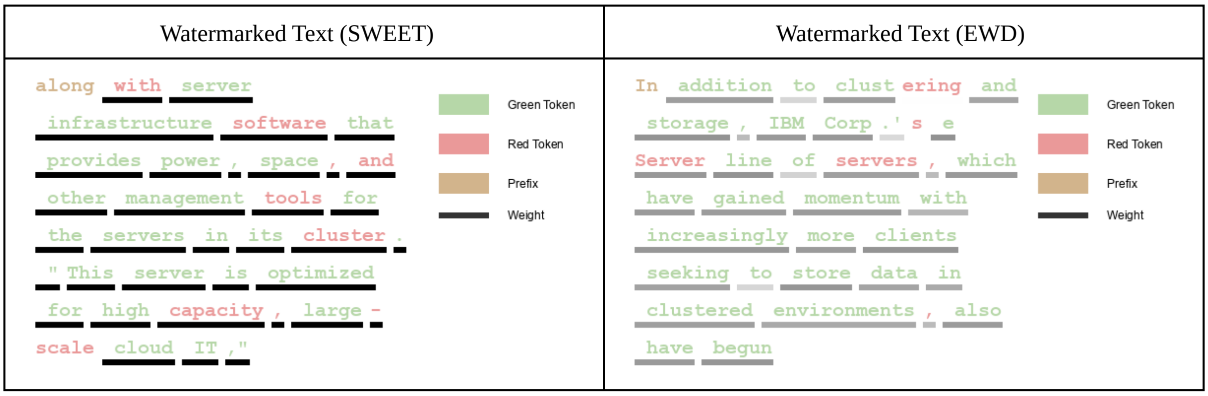

Figure 6: Visualization results of watermarked text using SWEET and EWD as watermarking algorithms.

1. On the other hand, the EWD method follows the KGW method during the watermarking process but assigns a continuous range of weights from 0 to 1 to each token based on its entropy during detection. Tokens with higher entropy receive higher weights, which helps alleviate the decline in detection accuracy in low-entropy environments. For the EWD method, the weights for each token are continuous values within the range of 0 to 1.

Figure 6 displays the visualization results for the SWEET and EWD methods. It can be observed that different shades of gray underline are used to represent the weight each token holds during detection. By visualizing the weights assigned to each token, users can gain a clearer insight into the internal mechanisms of watermarking algorithms that involve weight considerations. This enhanced visibility helps in understanding how these algorithms adapt to various textual contexts and the rationale behind the differential treatment of tokens based on their entropy or other characteristics.

C Scripts for Evaluation

In the evaluation module of MarkLLM, we have included a suite of Python scripts that are specifically designed to leverage the toolkit's pipeline for assessing the performance of various watermarking algorithms. These scripts are accessible within the **repository** under the directory *evaluation/examples/*. The files provided are *assess_detectability.py*, *assess_robustness.py*, and *assess_quality.py*. Utilizing the shells below (Listing 1, Listing 2, Listing 3), researchers can seamlessly conduct a thorough evaluation of various watermarking algorithms. This streamlined approach facilitates the comparison and analysis essential for advancing the field.

```
python evaluation/examples/assess_detectability.py --algorithm KGW --labels TPR F1
    --rules target_fpr --target_fpr 0.01

python evaluation/examples/assess_detectability.py --algorithm KGW --labels TPR TNR
    FPR FNR P R F1 ACC --rules best
```

Listing 1: Execution of assess detectability script. It accepts three command-line parameters: the name of the watermark algorithm to be evaluated, a list of metrics to return (choose one or more from FPR, FNR, TPR, TNR, P, R, F1, ACC), and the evaluation rule (choose from *best* and *target_fpr*). Additionally, *target_fpr* should be specified if the *target_fpr* evaluation rule is selected.

```
python evaluation/examples/assess_robustness.py --algorithm KGW --attack 'Word-D'

python evaluation/examples/assess_robustness.py --algorithm Unigram --attack '
    Doc-P(GPT-3.5)'
```

Listing 2: Execution of assess robustness script. It accepts two command-line parameters: the name of the watermark algorithm to be evaluated, and the name of the text tampering attack to be used (choose one from *Word-D*, *Word-S*, *Word-S(Context)*, *Doc-P(GPT-3.5)* and *Doc-P(Dipper)*).

```
python evaluation/examples/assess_quality.py --algorithm KGW --metric PPL
```

```
python evaluation/examples/assess_quality.py --algorithm SIR --metric 'Log
    Diversity'
```

Listing 3: Execution of assess quality script. It accepts two command-line parameters: the name of the watermark algorithm to be evaluated, and the name of the text quality metric to be used (choose one from *PPL*, *Log Diversity*, *BLEU*, *pass@1* and *GPT-4 Judge*).

D Hyper-parmaters for Experiment

D.1 Configuration Files of Watermarking Algorithms

```
{
    "algorithm_name": "KGW",
    "gamma": 0.5,
    "delta": 2.0,
    "hash_key": 15485863,
    "prefix_length": 1,
    "z_threshold": 4.0
}
```

Listing 4: configuration file, KGW.json

```
{
    "algorithm_name": "Unigram",
    "gamma": 0.5,
    "delta": 2.0,
    "hash_key": 15485863,
    "z_threshold": 4.0
}
```

Listing 5: configuration file, Unigram.json

```
{
    "algorithm_name": "SWEET",
    "gamma": 0.5,
    "delta": 2.0,
    "hash_key": 15485863,
    "z_threshold": 4.0,
    "prefix_length": 1,
    "entropy_threshold": 0.9
}
```

Listing 6: configuration file, SWEET.json

```
{
    "algorithm_name": "UPV",
    "gamma": 0.5,
    "delta": 2.0,
    "z_threshold": 4.0,
    "prefix_length": 1,
    "bit_number": 16,
    "sigma": 0.01,
    "default_top_k": 20,
    "generator_model_name": "watermark/upv/model/generator_model_b16_p1.pt",
    "detector_model_name": "watermark/upv/model/detector_model_b16_p1_z4.pt",
    "detect_mode": "network"
}
```

Listing 7: configuration file, UPV.json

```
{
    "algorithm_name": "EWD",
    "gamma": 0.5,
    "delta": 2.0,
    "hash_key": 15485863,
```

```
    "prefix_length": 1,
    "z_threshold": 4.0
}
```

Listing 8: configuration file, EWD.json

```
{
    "algorithm_name": "SIR",
    "delta": 1.0,
    "chunk_length": 10,
    "scale_dimension": 300,
    "z_threshold": 0.0,
    "transform_model_input_dim": 1024,
    "transform_model_name": "watermark/sir/model/transform_model_cbert.pth",
    "embedding_model_path": "watermark/sir/model/compositional-bert-large-uncased/",
    "mapping_name": "watermark/sir/mapping/300_mapping_50272.json"
}
```

Listing 9: configuration file, SIR.json

```
{
    "algorithm_name": "XSIR",
    "delta": 1.0,
    "chunk_length": 10,
    "scale_dimension": 300,
    "z_threshold": 0.2,
    "transform_model_input_dim": 768,
    "dictionary": "watermark/xsir/dictionary/dictionary.txt",
    "transform_model_name": "watermark/xsir/model/transform_model_x-sbert_10K.pth",
    "embedding_model_path": "watermark/xsir/model/paraphrase-multilingual-mpnet-base-v2",
    "mapping_name": "watermark/xsir/mapping/300_mapping_opt_1_3b.json"
}
```

Listing 10: configuration file, XSIR.json

```
{
    "algorithm_name": "EXP",
    "prefix_length": 4,
    "hash_key": 15485863,
    "threshold": 2.0,
    "sequence_length": 200
}
```

Listing 11: configuration file, EXP.json

```
{
    "algorithm_name": "EXPEdit",
    "pseudo_length": 420,
    "sequence_length": 200,
    "n_runs": 100,
    "key": 42,
    "p_threshold": 0.2
}
```

Listing 12: configuration file, EXP_Edit.json

D.2 Hyper-paramters of Evaluation Tools

Table 6 listed the hyper-parameters of evaluation tools used in Section 5.

E Comparison with Competitors

With the flourishing advancement of LLM watermarking technology, there has been a notable rise in the development of frameworks dedicated to this field. Among these, WaterBench (Tu et al., 2023) and Mark My Words (Piet et al., 2023) stand out as prominent examples. WaterBench primarily focuses on assessing

Table 6: Hyperparamers of evaluation tools used in experiment.

Tools	Hyperparameters
FundamentalSuccessRateCalculator	None
DynamicThresholdSuccessRateCalculator	None
WordDeletion	ratio: 0.3
SynonymSubstitution	ratio: 0.5
ContextAwareSynonymSubstitution	ratio: 0.5
GPTParaphraser	openai_model: "gpt-3.5-turbo", prompt: "Please rewrite the following text: "
DipperParaphraser	lex_diversity: 60, order_diversity: 0, max_new_tokens: 100, do_sample: True, top_p: 0.75
PPLCalculator	model: Llama-2-13B, tokenizer: Llama-2-13B
LogDiversityAnalyzer	None
BLEUCalculator	None
PassOrNotJudger	None
GPTTextDiscriminator	openai_model: "gpt-4", task_description: "Translate the following German text to English."

the impact of KGW (Kirchenbauer et al., 2023), Unigram (Zhao et al., 2024), and KGW-v2 (Kirchenbauer et al., 2024) on text quality, while Mark My Words evaluates the performance of KGW (Kirchenbauer et al., 2023), EXP (Aaronson and Kirchner, 2022), Christ (Christ et al., 2023), and EXP-Edit (Kuditipudi et al., 2023) across multiple dimensions, including text quality, robustness against tampering, and number of tokens needed for detection.

While both of these frameworks focus primarily on benchmark construction, similar to the evaluation module in MarkLLM, MarkLLM sets itself apart by not only offering easy-to-use evaluation tools and automated pipelines that encompass the aforementioned assessment perspectives but also providing a unified implementation framework for watermarking algorithms and visualization tools for their underlying mechanisms. This enhances its utility as a comprehensive toolkit. The integration of these functionalities renders MarkLLM a more versatile and accessible resource, enabling convenient usage, understanding, evaluation, and selection of diverse watermarking algorithms by researchers and the broader community. This plays a crucial role in fostering consensus both within and beyond the field.