Journal of Machine Learning Research 27 (2026) 1-22 Submitted 10/25; Revised 5/26; Published 7/26

MarkDiffusion: An Open-Source Toolkit for Generative Watermarking of Latent Diffusion Models

Leyi Pan[1*] panly24@mails.tsinghua.edu.cn
Sheng Guan[1,2*] guansheng2022@bupt.edu.cn
Zheyu Fu[1*] fuzy23@mails.tsinghua.edu.cn
Luyang Si[1*] sily23@mails.tsinghua.edu.cn
Huan Wang[1] huan-wan23@mails.tsinghua.edu.cn
Zian Wang[1] arthurwzaa@gmail.com
Hanqian Li[5] hli994@connect.hkust-gz.edu.cn
Xuming Hu[5] xuminghu97@gmail.com
Irwin King[3] king@cuhk.edu.hk
Philip S. Yu[4] psyu@uic.edu
Aiwei Liu[1†] liuaiwei20@gmail.com
Lijie Wen[1†] wenlj@tsinghua.edu.cn
[1] *Tsinghua University* [2] *Beijing University of Posts and Telecommunications*
[3] *The Chinese University of Hong Kong* [4] *University of Illinois at Chicago*
[5] *The Hong Kong University of Science and Technology (Guangzhou)*

Editor: Alexandre Gramfort

Abstract

We introduce MarkDiffusion, an open-source Python toolkit for generative watermarking of latent diffusion models. It comprises three key components: a `unified implementation framework` for streamlined watermarking algorithm integration and user-friendly interfaces; a `mechanism visualization suite` that intuitively presents embedded and extracted watermark patterns to aid public understanding; and a `comprehensive evaluation module` offering standard implementations of 24 tools for assessing detectability, robustness, and output quality, plus 8 automated evaluation pipelines.[1] Through MarkDiffusion, we seek to assist researchers, enhance public awareness of and engagement with generative watermarking, help build consensus, and advance research and applications. Code is available at `https://github.com/THU-BPM/MarkDiffusion`.

Keywords: watermark, latent diffusion models, toolkit, evaluation, visualization

1. Introduction

With advances in diffusion models (Ho et al., 2020; Song et al., 2020), latent diffusion models (LDMs) such as Stable Diffusion (Rombach et al., 2021) can generate high-quality images and videos for applications including digital art and film production. However, these capabilities also create risks of misuse, including manipulation of public opinion and intellectual property infringement (Bird et al., 2023; Jaidka et al., 2025). This concern has driven the development of watermarking techniques, which embed imperceptible, al-

∗. Equal contribution. † Corresponding authors.
1. Counts reflect the initial release; see the repository for the latest version.

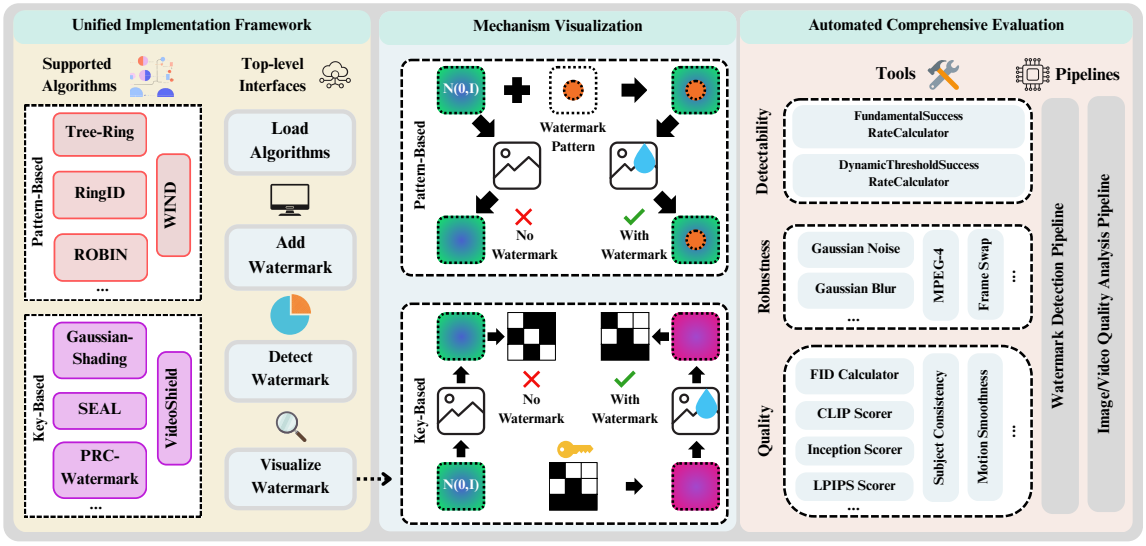

Figure 1: Architecture overview of MarkDiffusion.

gorithmically detectable signals to identify machine-generated content (Wen et al., 2023; Fernandez et al., 2023). In particular, generative watermarking injects signals into the latent space during LDM inference without model retraining or post-processing, making it the dominant approach. As an emerging technology, generative watermarking for LDMs faces key challenges: (i) incompatible, difficult-to-reuse algorithm implementations; (ii) technical complexity that impedes understanding and experimentation; and (iii) a lack of comprehensive evaluation frameworks. To address these challenges, we present MarkDiffusion, an open-source Python toolkit comprising:

- A unified implementation framework that seamlessly integrates eight state-of-the-art LDM watermarking algorithms through a modular design and easy-to-use APIs.
- A comprehensive evaluation module that integrates 24 tools and 8 automated pipelines, covering detectability, robustness, and output quality.
- An interactive visualization suite that intuitively demonstrates watermark embedding and detection mechanisms, enhancing accessibility for a wider audience.

2. Toolkit Design

2.1 Unified Implementation Framework

The implementation framework offers three key advantages.

Comprehensive Coverage. As shown in Table 1, our toolkit implements eight state-of-the-art watermarking algorithms from two major categories: (i) *pattern-based methods*, which embed predefined patterns (e.g., Fourier-domain rings) into the initial noise and detect them by measuring pattern similarity after inversion; and (ii) *key-based methods*, which use watermark keys to generate the initial noise and compare it with recovered noise for detection. The toolkit supports both image and video watermarking. Appendix A details the module and class designs.

Algorithms	Cat.	Target
Tree-Ring (Wen et al., 2023)	Pat.	Image
Ring-ID (Ci et al., 2024)	Pat.	Image
ROBIN (Huang et al., 2025)	Pat.	Image
WIND (Arabi et al., 2025a)	Pat.	Image
Gaussian-Shading (Yang et al., 2024)	Key	Image
PRC (Gunn et al., 2024)	Key	Image
SEAL (Arabi et al., 2025b)	Key	Image
VideoShield (Hu et al., 2025)	Key	Video

Table 1: Supported watermarking algorithms.

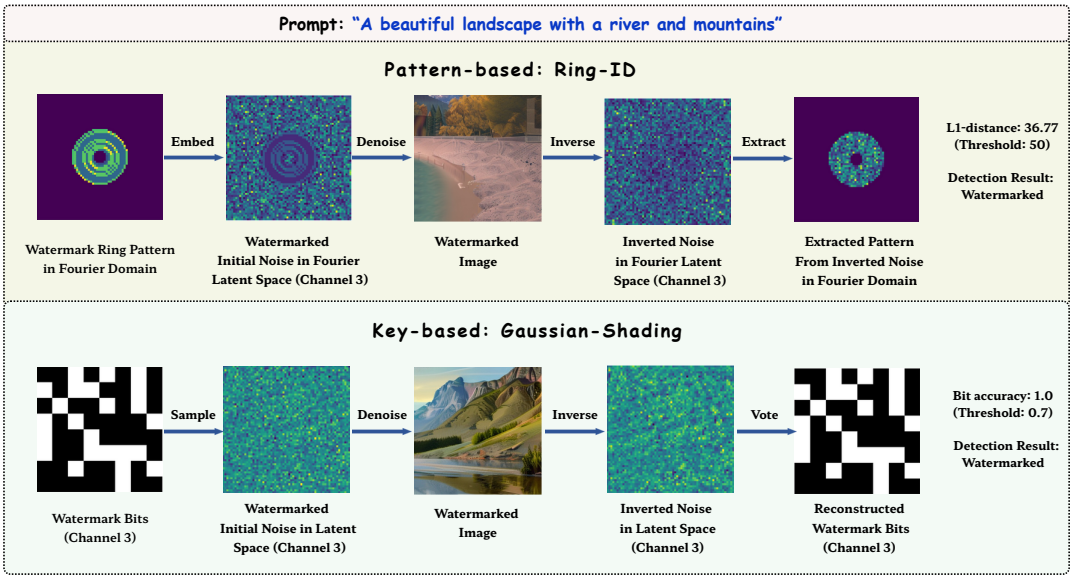

Figure 2: Visualization examples using MarkDiffusion.

Modular Extensibility. MarkDiffusion decouples the watermark pipeline into three independent modules: watermark embedding, watermark detection, and latent inversion. This modular design allows developers to extend the toolkit with new algorithms or customize existing components. For instance, users can plug in alternative detection metrics (e.g., p-values) or configure different inversion algorithms (e.g., DDIM Inversion (Song et al., 2020) or Exact Inversion (Hong et al., 2023)) without modifying the core framework.

User-Friendly Interface. All algorithms support watermark embedding and detection through direct calls to `generate_watermarked_media()` and `detect_watermark_in_media()`, respectively. Moreover, all algorithms are compatible with the `Diffusers` library. Appendix D provides comprehensive usage examples.

Perspective	Tools	Pipeline
Detectability	Fundamental SR. Calculator, Dynamic SR. Calculator	UWMDetect, WMDetect
Robustness (Image)	Rotation, Crop & Scale, Gaussian Noise, Gaussian Blur, JPEG, Color Jitter	UWMDetect, WMDetect
Robustness (Video)	MPEG-4, Frame Average, Frame Swap	UWMDetect, WMDetect
Quality (Image)	Inception Score, FID, LPIPS, CLIP-I, CLIP-T, PSNR, SSIM, NIQE	GroupQual, RepeatQual, RefQual, CompQual, DirectQual
Quality (Video)	Subject Consistency, Background Consistency, Motion Smoothness, Dynamic Degree, Imaging Quality	VideoQual

Table 2: Taxonomy of the evaluation module in MarkDiffusion.

2.2 Visualization Suite

MarkDiffusion provides tailored visualization modules for each algorithm, showing how watermarks are embedded, extracted, and verified. As illustrated in Figure 2, (i) for pattern-

based methods such as Ring-ID, the visualization shows that the embedded and extracted patterns both have a ring shape and that their ℓ_1 distance is below the preset threshold; and (ii) for key-based methods such as Gaussian-Shading, the suite displays the watermark key, the sampled initial noise derived from it, and the extracted watermark bits, highlighting their similarity. Appendix F provides additional examples.

2.3 Evaluation Module

Evaluating watermarking algorithms for LDMs requires assessment along three essential dimensions: (1) **watermark detectability**, or the ability to reliably distinguish watermarked from non-watermarked content; (2) **robustness against attacks**, measured after common manipulations such as compression, cropping, or noise addition; and (3) **impact on media quality**. To support these assessments, MarkDiffusion provides standard implementations of 24 evaluation tools and 8 automated pipelines that integrate watermarked-content generation, manipulation, watermark extraction, and metric computation into seamless workflows, as detailed in Table 2.

3. Experiments

We evaluate how well MarkDiffusion reproduces published experimental results. All experiments use Stable Diffusion v2.1 (Rombach et al., 2021) for image generation and ModelScope (Wang et al., 2023) for video generation. For images, we use 200 prompts from the Stable Diffusion Prompt data set (Gustavosta, 2023); video prompts are drawn from VBench (Huang et al., 2023). In each entry of Table 3, our result appears first and the result from the official implementation appears second. The close agreement demonstrates that our toolkit reliably reproduces the reported results.

Algorithms	**Detectability (↑) (TPR@FPR=1%)**	**Robustness (F1 score ↑)**			**Quality**	
		JPEG	**Gaussian Blur**	**Gaussian Noise**	**FID (↓)**	**CLIP-T (↑)**
Tree-Ring	1.00 / 1.00	0.85 / 0.88	0.66 / 0.68	0.95 / 0.93	32.30 / 33.15	0.658 / 0.665
Ring-ID	1.00 / 1.00	0.98 / 1.00	0.66 / 0.68	1.00 / 1.00	37.24 / 38.17	0.658 / 0.663
ROBIN	1.00 / 1.00	0.99 / 1.00	1.00 / 1.00	0.98 / 1.00	23.15 / 25.23	0.655 / 0.660
G-Shading	1.00 / 1.00	0.98 / 0.98	0.76 / 0.82	0.93 / 0.95	35.97 / 36.23	0.677 / 0.682
PRC	1.00 / 1.00	0.97 / 0.98	0.02 / 0.00	1.00 / 1.00	35.94 / 36.43	0.660 / 0.665
WIND	1.00 / 1.00	1.00 / 0.99	1.00 / 1.00	1.00 / 0.98	41.50 / 37.91	0.650 / 0.659
SEAL	0.95 / 0.98	0.79 / 0.81	0.67 / 0.67	0.68 / 0.67	37.35 / 36.97	0.659 / 0.664
Algorithms	**Detectability (↑)**	**MPEG-4**	**Frame Average**	**Frame Swap**	**Avg. of 5 Quality Metrics (↑)**	
VideoShield	1.00 / 1.00	0.98 / 0.98	0.99 / 1.00	1.00 / 1.00	0.803 / 0.804	

Table 3: Evaluation results of watermarking algorithms across key metrics.

4. Conclusion

This paper introduces MarkDiffusion, an open-source toolkit for generative watermarking of latent diffusion models. MarkDiffusion offers eight state-of-the-art watermarking algorithms through a unified, modular, and extensible interface. It also includes a visualization suite and an evaluation module with 24 tools and 8 automated pipelines, enabling researchers to compare algorithms and conduct systematic experiments.

Acknowledgments

The authors thank all contributors to the MarkDiffusion project and the open-source community for their valuable feedback. This work is primarily supported by the Key Research and Development Program of China (No. 2024YFB3309702).

Appendix A. Module and Class Design

This section details the system architecture, core modules, and class design principles underlying the toolkit.

A.1 Unified Implementation Framework

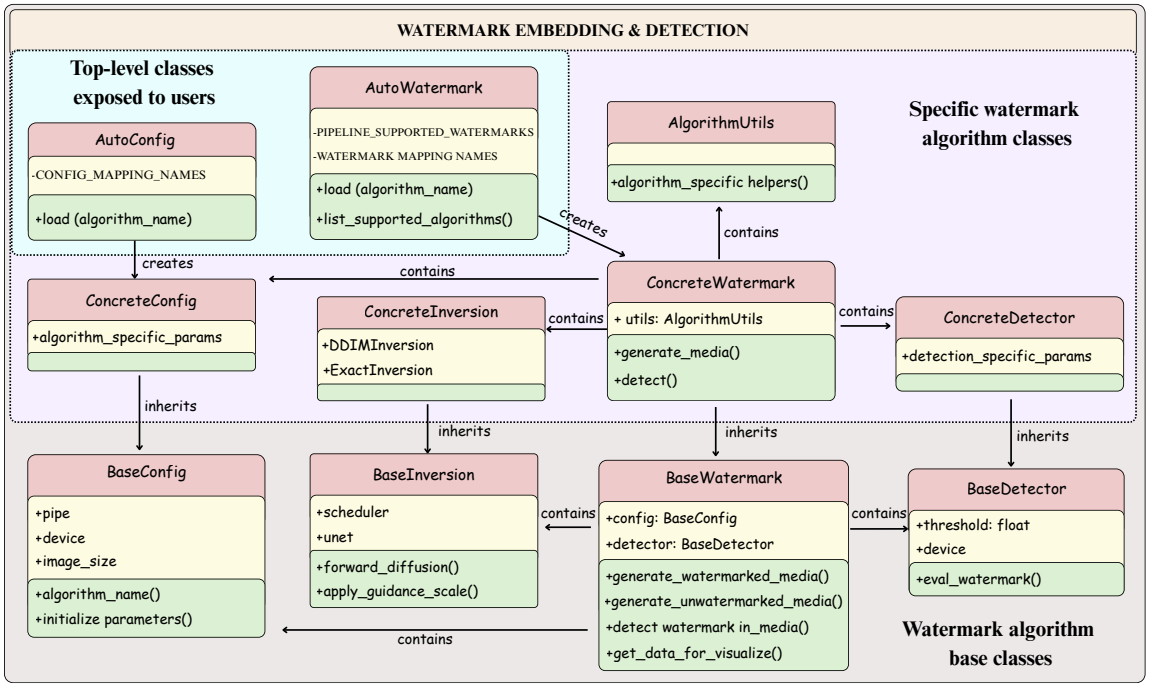

Figure 3: Detailed architecture of the unified implementation framework.

Figure 3 illustrates the unified implementation framework in MarkDiffusion, which handles watermark embedding and detection for every algorithm. The framework has a modular design comprising three layers:

- **Top-level classes exposed to users:** We provide two top-level classes, `AutoConfig` and `AutoWatermark`. Through the `load()` function, users specify an algorithm name to load the corresponding algorithm class (denoted by `ConcreteWatermark`) and configuration class (denoted by `ConcreteConfig`), as illustrated in the figure.
- **Specific watermark algorithm classes:** Each algorithm has a core watermark class (denoted by `ConcreteWatermark`). Its data members include `Config`, `Utils`,

`Inversion`, and `Detector` objects, each with a distinct role. The top-level interface provides `generate_watermarked_media()` and `detect_watermark_in_media()`, enabling watermark embedding and detection with a single line of code.

- **Watermark algorithm base classes:** These fundamental low-level classes serve as the base classes for all specific watermark algorithm classes and their components.

A.2 Mechanism Visualization

The visualization module facilitates understanding of watermarking mechanisms through algorithm-specific visualizers, as shown in Figure 4. The visualization classes obtain `DataForVisualization` objects from the algorithm class via the `get_data_for_visualization()` method. Each specific visualizer (created automatically by the `load()` method of `AutoVisualizer`) inherits from `BaseVisualizer` and implements methods for displaying latent representations, watermark patterns, and frequency-domain characteristics.

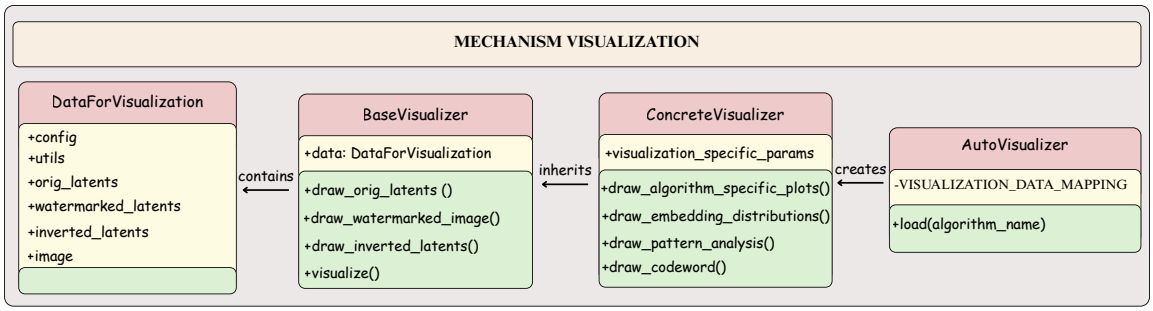

Figure 4: Detailed architecture of the mechanism visualization module.

A.3 Evaluation Module

Figure 5 details the evaluation module, which consists of eight pipelines covering three perspectives: detectability, robustness, and media quality. The `DetectionPipeline` evaluates detectability and robustness, while multiple `QualityPipeline` instances assess media quality. Each pipeline can integrate various toolsets.

A.4 Class Design Principles

The toolkit architecture is built upon well-established object-oriented design principles, incorporating key design patterns to ensure flexibility, consistency, and maintainability:

- **Template Method Pattern**: The `BaseWatermark` class defines the algorithmic framework for watermark operations. Subclasses implement specific steps, ensuring a consistent overall structure while allowing customization of individual components.
- **Strategy Pattern**: Detection algorithms and inversion methods are implemented as interchangeable strategies. This design enables dynamic selection and comparison of different approaches at runtime, fostering modularity and extensibility.

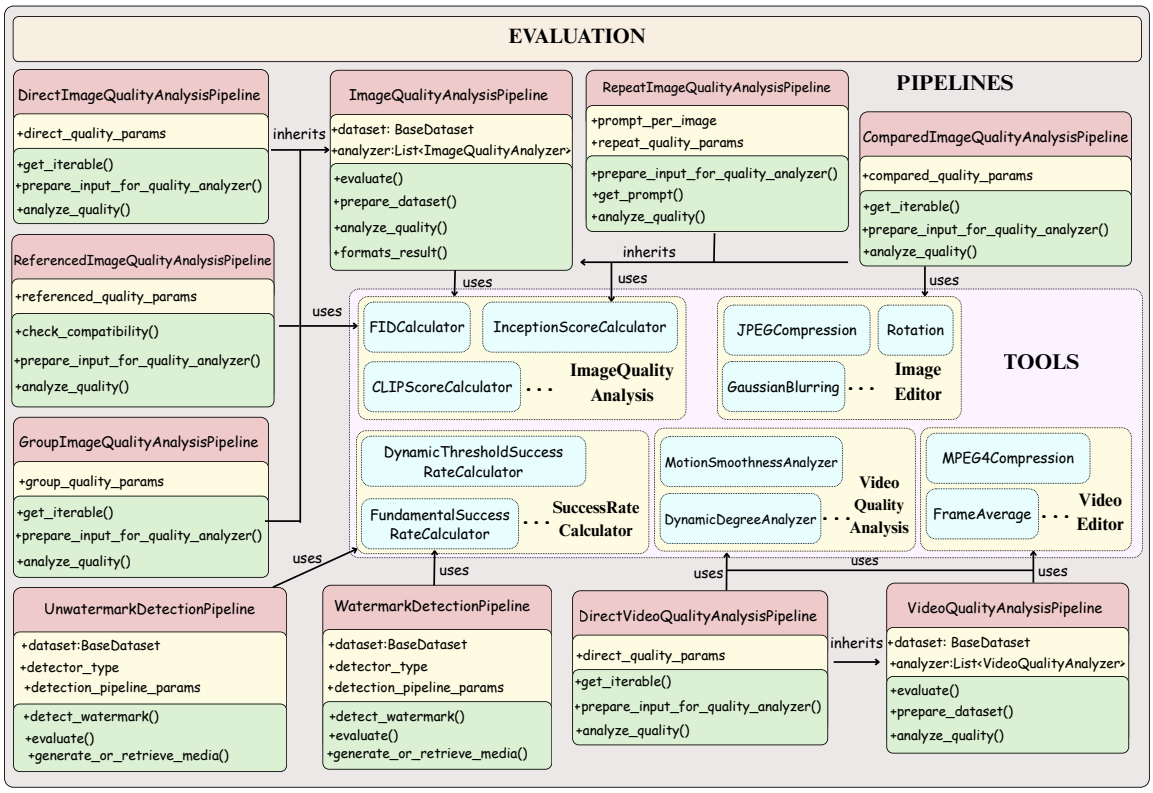

Figure 5: Detailed structure of the evaluation module.

- **Factory Pattern**: Both `AutoWatermark` and `AutoVisualizer` implement factory mechanisms to automatically instantiate algorithm-specific classes based on string identifiers. This simplifies object creation and reduces code coupling.
- **Composition Over Inheritance**: Complex functionalities are achieved by composing simpler, reusable components rather than relying on deep inheritance hierarchies. This approach enhances code maintainability, improves testing efficiency, and avoids the limitations of rigid inheritance structures.

Appendix B. Background: Generative Watermarking and Post-Hoc Watermarking

Existing watermarking techniques for diffusion models can broadly be divided into two categories: *generative watermarking*, which integrates watermarking signals directly into the generation process, and *post-hoc watermarking*, which modifies an image or video after it has been generated.

Post-hoc watermarking methods typically use traditional digital watermarking techniques, particularly frequency-domain approaches. These methods alter frequency-domain representations through transformations such as the Discrete Wavelet Transform (DWT) (Xia et al., 1998) or Discrete Cosine Transform (DCT) (Cox et al., 2008). One practical exam-

ple is the DwtDct Watermarking method (Navas et al., 2008), which has been applied in the open-source Stable Diffusion model. Frequency-domain techniques are robust against operations such as cropping, resizing, and compression.

In addition to these traditional methods, deep encoder–decoder frameworks, such as HiDDeN (Zhu et al., 2018), adopt an end-to-end approach to embedding and extracting watermarks. RivaGAN (Zhang et al., 2019) extends this framework through adversarial training, incorporating perturbations and image-processing operations into model training to enhance robustness. However, recent studies indicate that this watermarking paradigm is vulnerable to regeneration attacks using advanced generative models (Zhao et al., 2024). Furthermore, post-hoc methods introduce pixel-level perturbations that inevitably degrade the quality of generated images or videos, making undetectable watermarks difficult to maintain.

Conversely, generative watermarking methods embed signals directly within the image or video generation process. By avoiding direct modifications to the generated content, these approaches minimize perceptual impact (Arabi et al., 2025b) while improving robustness against regeneration attacks. Early generative watermarking techniques incorporated watermarks by fine-tuning specific model components, such as the decoder of a latent diffusion model. A representative example is Stable Signature (Fernandez et al., 2023), which fine-tunes the latent decoder to embed a binary signature that a lightweight detector can extract from generated images. Although the per-key fine-tuning cost is modest (on the order of a minute on a single GPU) and detection is efficient (tens of milliseconds per image), this paradigm couples each watermark key to a distinct set of fine-tuned model weights. Consequently, supporting multiple users or keys requires maintaining and switching between separate model checkpoints, limiting flexibility in deployments that require frequent key rotation or many distinct keys.

An alternative approach modifies latent representations at inference time, allowing watermarks to be embedded in generated images or videos without additional training or direct manipulation of the generated content. To detect such watermarks, diffusion inversion algorithms, such as DDIM Inversion (Song et al., 2020), reconstruct the latent representations to determine whether the watermark key remains detectable. Tree-Ring (Wen et al., 2023), for example, embeds a Fourier-domain ring pattern in the initial latent space for watermarked image generation. It detects the watermark by calculating the ℓ_1 distance between the reconstructed and original patterns in Fourier space.

MarkDiffusion builds upon this emerging class of generative watermarking techniques by introducing a comprehensive toolkit for generating, detecting, and evaluating such watermarks. It offers a flexible, modular platform that supports various generative watermarking algorithms, enabling research and experimentation with this technology for latent diffusion models.

Appendix C. Comparison with Existing Toolkits

To situate MarkDiffusion within the broader landscape of watermarking technologies, this section compares it with two relevant toolkits: **invisible-watermark** (ShieldMnt, 2021) and **MarkLLM** (Pan et al., 2024). This comparison clarifies the technical paradigm of

	Watermark Paradigm	Application Domain	Core Focus
invisible-watermark	Post-hoc	General image files	A lightweight tool for applying watermarks to existing images.
MarkLLM	Generative	Large language models (text)	A comprehensive toolkit for generative text watermarking.
MarkDiffusion	Generative	Diffusion models (images & video)	A comprehensive toolkit for generative watermarking in visual media.

Table 4: Comparison of MarkDiffusion with existing watermarking toolkits.

MarkDiffusion and highlights its contributions to generative media. Table 4 summarizes the core distinctions.

The primary distinction between MarkDiffusion and invisible-watermark lies in their watermarking paradigms. invisible-watermark is a post-hoc tool that applies watermarks to images after generation. In contrast, MarkDiffusion takes a generative, in-process approach, integrating watermarking directly into the synthesis pipeline of diffusion models for both images and videos. Thus, MarkDiffusion embeds provenance information at the point of creation rather than through a subsequent modification.

Although MarkDiffusion shares a generative paradigm with MarkLLM, their application domains are distinct. MarkLLM is a framework for watermarking content generated by large language models (LLMs), focusing exclusively on text (Kirchenbauer et al., 2023; Christ et al., 2024; Liu et al., 2024a, 2025; Pan et al., 2025; Liu et al., 2024b). MarkLLM nevertheless provided conceptual inspiration for MarkDiffusion: its Python package structure and evaluation-pipeline design served as architectural references. MarkDiffusion adapts and extends these principles to meet the challenges of the visual domain, bridging the gap between generative text-watermarking concepts and their practical implementation in image and video synthesis.

In summary, MarkDiffusion complements these tools by addressing a distinct need. It differs from the post-hoc methodology of invisible-watermark through its in-process approach and adapts MarkLLM's generative watermarking philosophy from text to visual media, offering a specialized and comprehensive toolkit for diffusion models.

Appendix D. Usage Examples

The following code snippets illustrate how to use MarkDiffusion in a project.

D.1 Generating and Detecting Watermarked Media

```
# Load watermarking algorithm
mywatermark = AutoWatermark.load(
    'GS',
    algorithm_config=f'config/GS.json',
    diffusion_config=diffusion_config
)
# Generate watermarked image
watermarked_image = mywatermark.generate_watermarked_media(
```

```
    input_data="A beautiful landscape with a river and mountains"
)
# Visualize the watermarked image
watermarked_image.show()
# Detect watermark
detection_result = mywatermark.detect_watermark_in_media(watermarked_image)
print(detection_result)
```

D.2 Visualizing Watermarking Mechanisms

```
from visualize.auto_visualization import AutoVisualizer

# Get data for visualization
data_for_visualization = mywatermark.get_data_for_visualize(
    watermarked_image)

# Load Visualizer
visualizer = AutoVisualizer.load('GS', data_for_visualization=
    data_for_visualization)

# Draw diagrams on Matplotlib canvas
fig = visualizer.visualize(rows=2, cols=2, methods=['draw_watermark_bits', '
    draw_reconstructed_watermark_bits', 'draw_inverted_latents', '
    draw_inverted_latents_fft'])
```

D.3 Evaluation Examples

D.3.1 Watermark Detection Pipeline

```
# Dataset
my_dataset = StableDiffusionPromptsDataset(max_samples=200)

pipeline1 = WatermarkMediaDetectionPipeline(dataset=my_dataset,
    media_editor_list=[JPEGCompression(quality=60)],
show_progress=True, return_type=DetectionPipelineReturnType.SCORES)

pipeline2 = UnWatermarkMediaDetectionPipeline(datmy_aset=my_dataset,
    media_editor_list=[],
show_progress=True, return_type=DetectionPipelineReturnType.SCORES)

detection_kwargs = {
    "num_inference_steps": 50,
    "guidance_scale": 1.0,
}

calculator = DynamicThresholdSuccessRateCalculator(labels=labels, rule=rules
    , target_fpr=target_fpr, reverse=True)
print(calculator.calculate(pipeline1.evaluate(my__watermark,
    detection_kwargs=detection_kwargs), pipeline2.evaluate(my__watermark,
    detection_kwargs=detection_kwargs))))
```

D.3.2 Image Quality Analysis Pipeline

```
if metric == 'NIQE':
    my_dataset = StableDiffusionPromptsDataset(max_samples=max_samples)
    pipeline = DirectImageQualityAnalysisPipeline(dataset=my_dataset,
        watermarkMed_iae_editor_list=[],
        unwatermarked_image_editor_list=[],
        analyzers=[NIQECalculator()],
        show_progress=True,
        return_type=QualityPipelineReturnType.MEAN_SCORES)

elif metric == 'CLIP':
    my_dataset = MSCOCODataset(max_samples=max_samples)
    pipeline = ReferencedImageQualityAnalysisPipeline(dataset=my_dataset,
        watermarked_image_editor_list=[],
        unwatermarked_image_editor_list=[],
        analyzers=[CLIPScoreCalculator()],
        unwatermarked_image_source='generated',
        reference_image_source='natural',
        show_progress=True,
        return_type=QualityPipelineReturnType.MEAN_SCORES)

elif metric == 'FID':
    my_dataset = MSCOCODataset(max_samples=max_samples)
    pipeline = GroupImageQualityAnalysisPipeline(dataset=my_dataset,
        watermarked_image_editor_list=[],
        unwatermarked_image_editor_list=[],
        analyzers=[FIDCalculator()],
        unwatermarked_image_source='generated',
        reference_image_source='natural',
        show_progress=True,
        return_type=QualityPipelineReturnType.MEAN_SCORES)
elif metric == 'IS':
    my_dataset = StableDiffusionPromptsDataset(max_samples=max_samples)
    pipeline = GroupImageQualityAnalysisPipeline(dataset=my_dataset,
        watermarked_image_editor_list=[],
        unwatermarked_image_editor_list=[],
        analyzers=[InceptionScoreCalculator()],
        show_progress=True,
        return_type=QualityPipelineReturnType.MEAN_SCORES)

elif metric == 'LPIPS':
    my_dataset = StableDiffusionPromptsDataset(max_samples=10)
    pipeline = RepeatImageQualityAnalysisPipeline(dataset=my_dataset,
        prompt_per_image=20,
        watermarked_image_editor_list=[],
        unwatermarked_image_editor_list=[],
        analyzers=[LPIPSAnalyzer()],
        show_progress=True,
        return_type=QualityPipelineReturnType.MEAN_SCORES)

elif metric == 'PSNR':
    my_dataset = StableDiffusionPromptsDataset(max_samples=max_samples)
    pipeline = ComparedImageQualityAnalysisPipeline(dataset=my_dataset,
        watermarked_image_editor_list=[],
        unwatermarked_image_editor_list=[],
        analyzers=[PSNRAnalyzer()],
```

```
        show_progress=True,
        return_type=QualityPipelineReturnType.MEAN_SCORES)
else:
    raise ValueError('Invalid metric')


my_watermark = AutoWatermark.load(f'{algorithm_name}',
    algorithm_config=f'config/{algorithm_name}.json',
    diffusion_config=diffusion_config)
print(pipeline.evaluate(my_watermark))
```

D.3.3 Video Quality Analysis Pipeline

```
from evaluation.tools.video_quality_analyzer import (
    SubjectConsistencyAnalyzer,
    MotionSmoothnessAnalyzer,
    DynamicDegreeAnalyzer,
    BackgroundConsistencyAnalyzer,
    ImagingQualityAnalyzer
)

# Load VBench dataset
my_dataset = VBenchDataset(max_samples=200, dimension=dimension)

# Initialize analyzer based on metric
if metric == 'subject_consistency':
    analyzer = SubjectConsistencyAnalyzer(device=device)
elif metric == 'motion_smoothness':
    analyzer = MotionSmoothnessAnalyzer(device=device)
elif metric == 'dynamic_degree':
    analyzer = DynamicDegreeAnalyzer(device=device)
elif metric == 'background_consistency':
    analyzer = BackgroundConsistencyAnalyzer(device=device)
elif metric == 'imaging_quality':
    analyzer = ImagingQualityAnalyzer(device=device)
else:
    raise ValueError(f'Invalid metric: {metric}. Supported metrics:
        subject_consistency, motion_smoothness, dynamic_degree,
        background_consistency, imaging_quality')

# Create video quality analysis pipeline
pipeline = DirectVideoQualityAnalysisPipeline(
    dataset=my_dataset,
    watermarked_video_editor_list=[],
    unwatermarked_video_editor_list=[],
    watermarked_frame_editor_list=[],
    unwatermarked_frame_editor_list=[],
    analyzers=[analyzer],
    show_progress=True,
    return_type=QualityPipelineReturnType.MEAN_SCORES
)

print(pipeline.evaluate(my_watermark))
```

Algorithm Name	Category	Methodology
Tree-Ring (Wen et al., 2023)	Pattern-Based	Embeds a concentric-ring pattern in the Fourier domain of the initial latent; detection inverts the generated image to its latent representation and measures its distance from the original pattern.
Ring-ID (Ci et al., 2024)	Pattern-Based	Building on Tree-Ring, it imprints multiple keys in different channels to improve watermark verification and multi-key identification.
ROBIN (Huang et al., 2025)	Pattern-Based	Building on Tree-Ring, it embeds a pattern in an intermediate diffusion state and produces an optimized guidance signal during generation to minimize the impact on image semantics.
WIND (Arabi et al., 2025a)	Pattern-Based	Supports a large key space by maintaining initial-noise samples with different seeds, partitioning them into groups, and imprinting Fourier patterns as group identifiers.
Gaussian-Shading (Yang et al., 2024)	Key-Based	Selects a fixed quadrant of the latent space as the watermark key and repeats it to create a distortion-free initial latent for image generation.
PRC-Watermark (Gunn et al., 2024)	Key-Based	Uses pseudorandom error-correcting codes (Christ and Gunn, 2024) to create the initial latents for generation, thereby achieving computational undetectability.
SEAL (Arabi et al., 2025b)	Key-Based	Correlates the watermark with image semantics, using the semantic vector of the unwatermarked image and a secret salt to create the initial noise without requiring access to historical watermark-key databases.
VideoShield (Hu et al., 2025)	Key-Based	Extends the core idea of Gaussian-Shading to video generation.

Table 5: Details of watermarking algorithms implemented in our toolkit.

Appendix E. Details of the Integrated Watermarking Algorithms

Table 5 presents the eight watermarking algorithms integrated into MarkDiffusion. They span the two major paradigms of generative watermarking and cover both image and video generation. Each algorithm has distinct design objectives and innovations.

Appendix F. Additional Visualization Examples

This section provides visualization examples for each supported watermarking algorithm and the corresponding invocation code.

Tree-Ring Visualization:

```
from visualize.auto_visualization import AutoVisualizer

data_for_visualization = mywatermark.get_data_for_visualize(
    watermarked_image)

visualizer = AutoVisualizer.load('TR', data_for_visualization=
    data_for_visualization)

method_kwargs = [{},{"channel": 0}, {}, {"channel": 0}, {}]

fig = visualizer.visualize(
    rows=1,
    cols=5,
    methods=['draw_pattern_fft','draw_orig_latents_fft', '
        draw_watermarked_image', 'draw_inverted_latents_fft', '
        draw_inverted_pattern_fft'],
    method_kwargs=method_kwargs,
    save_path='TR_watermark_visualization.pdf'
    )
```

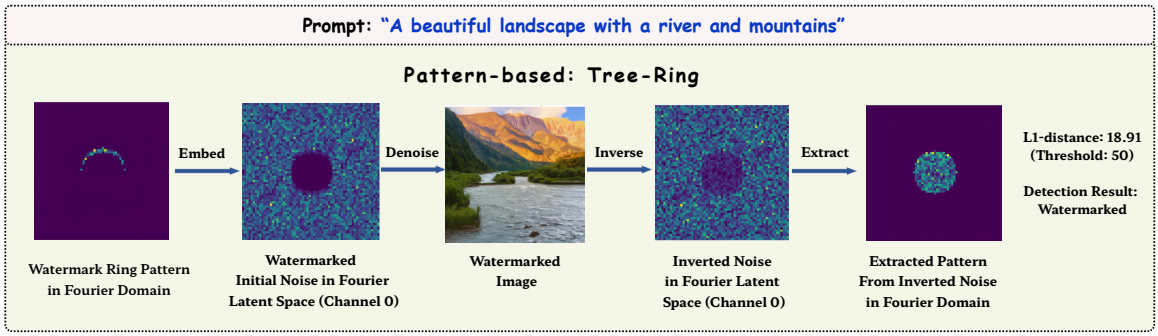

Figure 6: Example visualization of the Tree-Ring watermark.

Gaussian-Shading Visualization:

```
from visualize.auto_visualization import AutoVisualizer

data_for_visualization = mywatermark.get_data_for_visualize(
    watermarked_image)

visualizer = AutoVisualizer.load('GS', data_for_visualization=
    data_for_visualization)

method_kwargs = [{"channel": 0},{"channel": 0}, {}, {"channel": 0}, {"
    channel": 0}]

fig = visualizer.visualize(
    rows=1,
    cols=5,
    methods=[ 'draw_watermark_bits', 'draw_orig_latents', '
        draw_watermarked_image','draw_inverted_latents', '
        draw_reconstructed_watermark_bits'],
    method_kwargs=method_kwargs,
```

```
    save_path='GS_watermark_visualization.pdf'
    )
```

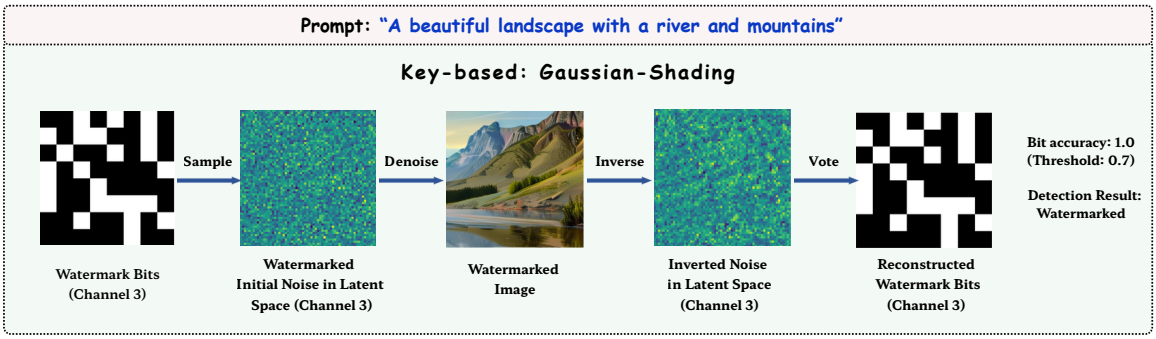

Figure 7: Example visualization of the Gaussian-Shading watermark.

Ring-ID Visualization:

```
from visualize.auto_visualization import AutoVisualizer

data_for_visualization = mywatermark.get_data_for_visualize(
    watermarked_image)

visualizer = AutoVisualizer.load('RI', data_for_visualization=
    data_for_visualization)

method_kwargs = [{}, {"channel": 3}, {}, {"channel": 3}, {}]
fig = visualizer.visualize(
    rows=1,
    cols=5,
    methods=['draw_ring_pattern_fft',
    'draw_orig_latents_fft', 'draw_watermarked_image',
    'draw_inverted_latents_fft',
    'draw_heter_pattern_fft'],
    method_kwargs=method_kwargs,
    save_path='RI_watermark_visualization.pdf'
)
```

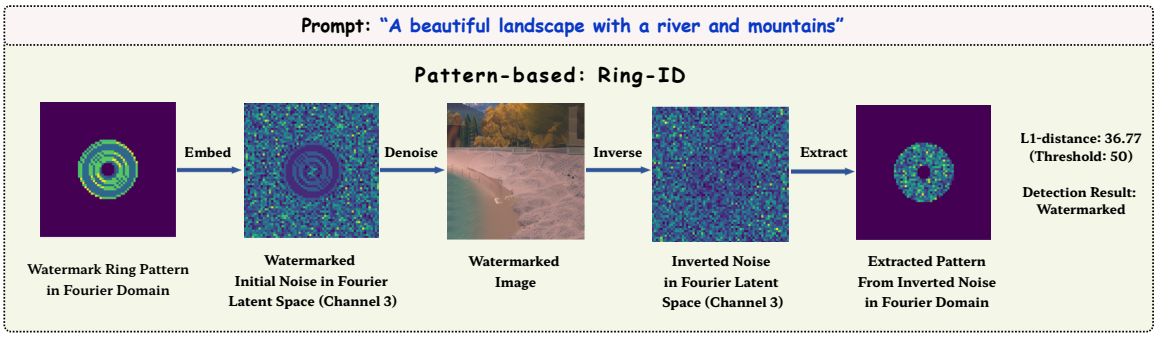

Figure 8: Example visualization of the Ring-ID watermark.

SEAL Visualization:

```
from visualize.auto_visualization import AutoVisualizer
data_for_visualization = mywatermark.get_data_for_visualize(
    watermarked_image)

visualizer = AutoVisualizer.load('SEAL', data_for_visualization=
    data_for_visualization)

method_kwargs = [{}, {"channel": 2}, {}, {"channel": 2}, {}]

fig = visualizer.visualize(
    rows=1,
    cols=5,
    methods=['draw_embedding_distributions','draw_orig_latents', '
        draw_watermarked_image', 'draw_inverted_latents', 'draw_patch_diff'
        ],
    method_kwargs=method_kwargs,
    save_path='SEAL_watermark_visualization.pdf'
    )
```

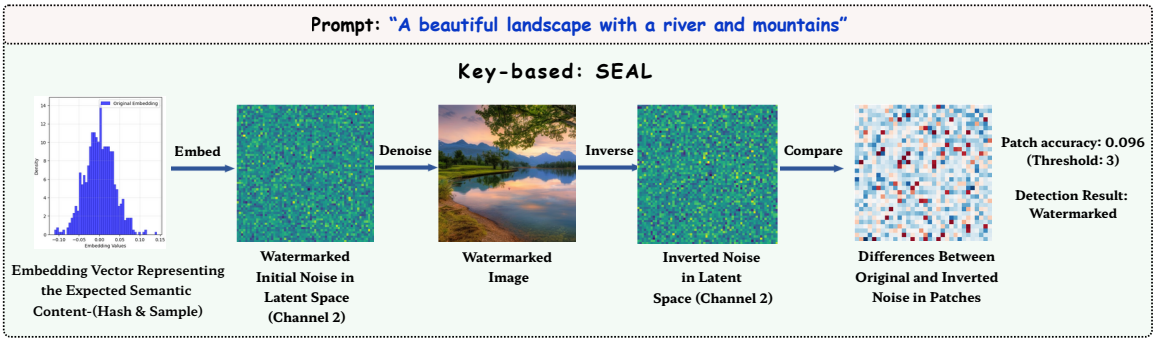

Figure 9: Example visualization of the SEAL watermark.

WIND Visualization:

```
from visualize.auto_visualization import AutoVisualizer

data_for_visualization = mywatermark.get_data_for_visualize(
    watermarked_image)

visualizer = AutoVisualizer.load('WIND', data_for_visualization=
    data_for_visualization)

method_kwargs = [{"channel": 2}, {"channel": 2}, {}, {"channel": 2}, {"
    channel": 2}]

fig = visualizer.visualize(
    rows=1,
    cols=5,
    methods=['draw_group_pattern_fft', 'draw_orig_latents_fft', '
        draw_watermarked_image', 'draw_inverted_latents_fft', '
        draw_inverted_group_pattern_fft'],
    method_kwargs=method_kwargs,
```

```
    save_path='WIND_watermark_visualization.pdf'
    )
```

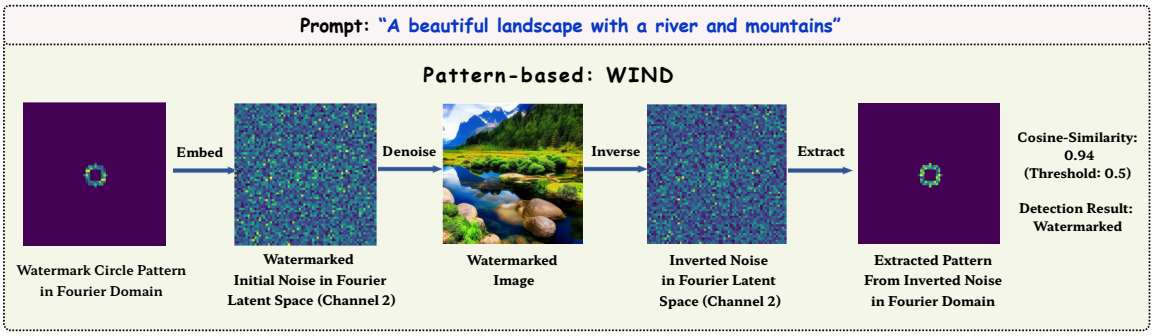

Figure 10: Example visualization of the WIND watermark.

ROBIN Visualization:

```
from visualize.auto_visualization import AutoVisualizer

data_for_visualization = mywatermark.get_data_for_visualize(
    watermarked_image)

visualizer = AutoVisualizer.load('ROBIN', data_for_visualization=
    data_for_visualization)

method_kwargs = [{}, {"channel": 3}, {}, {"channel": 3}, {}]

fig = visualizer.visualize(
    rows=1,
    cols=5,
    methods=['draw_pattern_fft', 'draw_orig_latents_fft', '
        draw_watermarked_image', 'draw_inverted_latents_fft', '
        draw_inverted_pattern_fft'],
    method_kwargs=method_kwargs,
    save_path='ROBIN_watermark_visualization.pdf'
    )
```

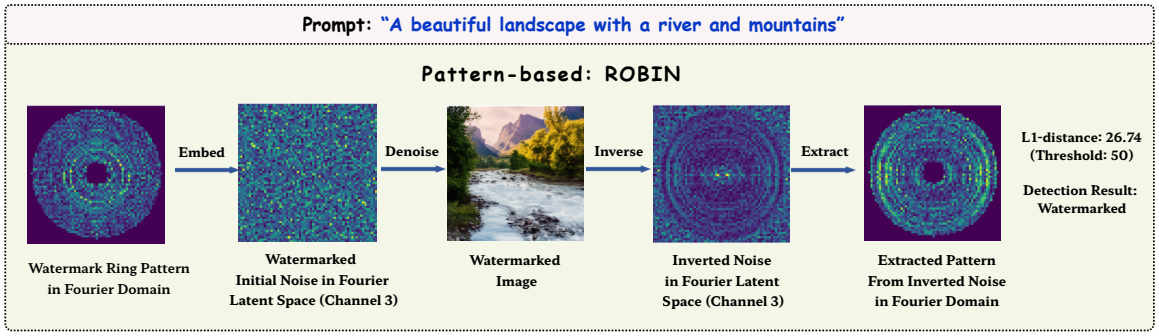

Figure 11: Example visualization of the ROBIN watermark.

PRC Visualization:

```
from visualize.auto_visualization import AutoVisualizer

data_for_visualization = mywatermark.get_data_for_visualize(
    watermarked_image)

visualizer = AutoVisualizer.load('PRC', data_for_visualization=
    data_for_visualization)

method_kwargs = [{}, {"channel": 3}, {}, {"channel": 3}, {}]

fig = visualizer.visualize(
    rows=1,
    cols=5,
    methods=['draw_codeword','draw_orig_latents', 'draw_watermarked_image',
        'draw_inverted_latents','draw_recovered_codeword'],
    method_kwargs=method_kwargs,
    save_path='PRC_watermark_visualization.pdf'
    )
```

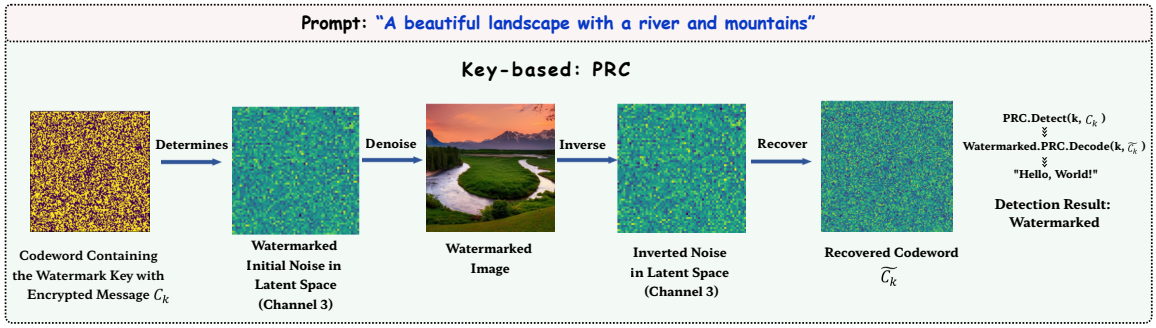

Figure 12: Example visualization of the PRC watermark.

VideoShield Visualization:

```
from visualize.auto_visualization import AutoVisualizer

data_for_visualization = mywatermark.get_data_for_visualize(
    video_frames=watermarked_video,
    prompt="",
    guidance_scale=1.0,
    num_inference_steps=25,
)
visualizer = AutoVisualizer.load('VideoShield', data_for_visualization=
    data_for_visualization)

method_kwargs = [{"channel": 1}, {"channel": 1}, {"num_frames": 4}, {"
    channel": 1}, {"channel": 1}]

fig = visualizer.visualize(
    rows=1,
    cols=5,
```

```
    methods=['draw_watermark_bits','draw_orig_latents', '
        draw_watermarked_video_frames', 'draw_inverted_latents', '
        draw_reconstructed_watermark_bits'],
    method_kwargs=method_kwargs,
    save_path='VideoShield_watermark_visualization.pdf'
    )
```

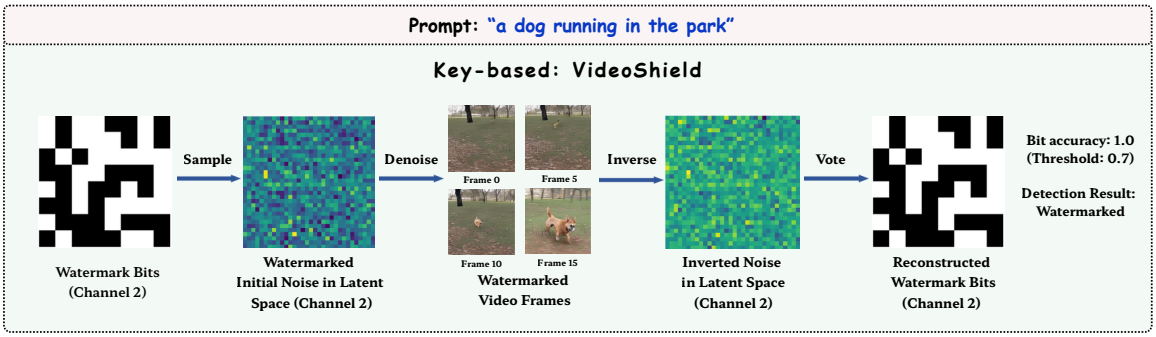

Figure 13: Example visualization of the VideoShield watermark.

References

Kasra Arabi, Benjamin Feuer, R. Teal Witter, Chinmay Hegde, and Niv Cohen. Hidden in the Noise: Two-Stage Robust Watermarking for Images, February 2025a.

Kasra Arabi, R. Teal Witter, Chinmay Hegde, and Niv Cohen. SEAL: Semantic Aware Image Watermarking, April 2025b.

Charlotte Bird, Eddie Ungless, and Atoosa Kasirzadeh. Typology of risks of generative text-to-image models. In *Proceedings of the 2023 AAAI/ACM Conference on AI, Ethics, and Society*, pages 396–410, 2023.

Miranda Christ and Sam Gunn. Pseudorandom Error-Correcting Codes, 2024.

Miranda Christ, Sam Gunn, and Or Zamir. Undetectable watermarks for language models. In Shipra Agrawal and Aaron Roth, editors, *Proceedings of Thirty Seventh Conference on Learning Theory*, volume 247 of *Proceedings of Machine Learning Research*, pages 1125–1139. PMLR, 30 Jun–03 Jul 2024. URL `https://proceedings.mlr.press/v247/christ24a.html`.

Hai Ci, Pei Yang, Yiren Song, and Mike Zheng Shou. RingID: Rethinking Tree-Ring Watermarking for Enhanced Multi-Key Identification, July 2024.

Ingemar J Cox, Matthew L Miller, Jeffrey A Bloom, Jessica Fridrich, and Ton Kalker. Digital watermarking. *Morgan Kaufmann Publishers*, 54:56–59, 2008.

Pierre Fernandez, Guillaume Couairon, Hervé Jégou, Matthijs Douze, and Teddy Furon. The stable signature: Rooting watermarks in latent diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 22466–22477, 2023.

Sam Gunn, Xuandong Zhao, and Dawn Song. An undetectable watermark for generative image models, October 2024.

Gustavosta. Gustavosta/stable-diffusion-prompts, 2023. URL `https://huggingface.co/datasets/Gustavosta/Stable-Diffusion-Prompts`.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *arXiv preprint arxiv:2006.11239*, 2020.

Seongmin Hong, Kyeonghyun Lee, Suh Yoon Jeon, Hyewon Bae, and Se Young Chun. On Exact Inversion of DPM-Solvers, November 2023.

Runyi Hu, Jie Zhang, Yiming Li, Jiwei Li, Qing Guo, Han Qiu, and Tianwei Zhang. VideoShield: Regulating Diffusion-based Video Generation Models via Watermarking, February 2025.

Huayang Huang, Yu Wu, and Qian Wang. ROBIN: Robust and Invisible Watermarks for Diffusion Models with Adversarial Optimization, April 2025.

Ziqi Huang, Yinan He, Jiashuo Yu, Fan Zhang, Chenyang Si, Yuming Jiang, Yuanhan Zhang, Tianxing Wu, Qingyang Jin, Nattapol Chanpaisit, Yaohui Wang, Xinyuan Chen, Limin Wang, Dahua Lin, Yu Qiao, and Ziwei Liu. VBench: Comprehensive Benchmark Suite for Video Generative Models, November 2023.

Kokil Jaidka, Tsuhan Chen, Simon Chesterman, Wynne Hsu, Min-Yen Kan, Mohan Kankanhalli, Mong Li Lee, Gyula Seres, Terence Sim, Araz Taeihagh, et al. Misinformation, disinformation, and generative ai: Implications for perception and policy. *Digital Government: Research and Practice*, 6(1):1–15, 2025.

John Kirchenbauer, Jonas Geiping, Yuxin Wen, Jonathan Katz, Ian Miers, and Tom Goldstein. A watermark for large language models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett, editors, *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, 2023. URL https://proceedings.mlr.press/v202/kirchenbauer23a.html.

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.

Aiwei Liu, Leyi Pan, Yijian Lu, Jingjing Li, Xuming Hu, Xi Zhang, Lijie Wen, Irwin King, Hui Xiong, and Philip Yu. A survey of text watermarking in the era of large language models. *ACM Comput. Surv.*, 57(2), November 2024b. ISSN 0360-0300. doi: 10.1145/3691626. URL https://doi.org/10.1145/3691626.

Aiwei Liu, Sheng Guan, Yiming Liu, Leyi Pan, Yifei Zhang, Liancheng Fang, Lijie Wen, Philip S. Yu, and Xuming Hu. Can watermarked LLMs be identified by users via crafted prompts? In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=ujpAYpFDEA.

KA Navas, Mathews Cheriyan Ajay, M Lekshmi, Tampy S Archana, and M Sasikumar. Dwt-dct-svd based watermarking. In *2008 3rd international conference on communication systems software and middleware and workshops (COMSWARE'08)*, pages 271–274. IEEE, 2008.

Leyi Pan, Aiwei Liu, Zhiwei He, Zitian Gao, Xuandong Zhao, Yijian Lu, Binglin Zhou, Shuliang Liu, Xuming Hu, Lijie Wen, Irwin King, and Philip S. Yu. MarkLLM: An open-source toolkit for LLM watermarking. In Delia Irazu Hernandez Farias, Tom Hope, and Manling Li, editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 61–71, Miami, Florida, USA, November 2024. Association for Computational Linguistics. URL https://aclanthology.org/2024.emnlp-demo.7.

Leyi Pan, Aiwei Liu, Shiyu Huang, Yijian Lu, Xuming Hu, Lijie Wen, Irwin King, and Philip S Yu. Can llm watermarks robustly prevent unauthorized knowledge distillation? *arXiv preprint arXiv:2502.11598*, 2025.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2021.

ShieldMnt. invisible-watermark, 2021. URL `https://github.com/ShieldMnt/invisible-watermark`.

Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv:2010.02502*, October 2020. URL `https://arxiv.org/abs/2010.02502`.

Jiuniu Wang, Hangjie Yuan, Dayou Chen, Yingya Zhang, Xiang Wang, and Shiwei Zhang. Modelscope text-to-video technical report. *arXiv preprint arXiv:2308.06571*, 2023.

Yuxin Wen, John Kirchenbauer, Jonas Geiping, and Tom Goldstein. Tree-Ring Watermarks: Fingerprints for Diffusion Images that are Invisible and Robust, July 2023.

Xiang-Gen Xia, Charles G Boncelet, and Gonzalo R Arce. Wavelet transform based watermark for digital images. *Optics Express*, 3(12):497–511, 1998.

Zijin Yang, Kai Zeng, Kejiang Chen, Han Fang, Weiming Zhang, and Nenghai Yu. Gaussian Shading: Provable Performance-Lossless Image Watermarking for Diffusion Models, May 2024.

Kevin Alex Zhang, Lei Xu, Alfredo Cuesta-Infante, and Kalyan Veeramachaneni. Robust invisible video watermarking with attention. *arXiv preprint arXiv:1909.01285*, 2019.

Xuandong Zhao, Kexun Zhang, Zihao Su, Saastha Vasan, Ilya Grishchenko, Christopher Kruegel, Giovanni Vigna, Yu-Xiang Wang, and Lei Li. Invisible image watermarks are provably removable using generative ai. *Advances in neural information processing systems*, 37:8643–8672, 2024.

Jiren Zhu, Russell Kaplan, Justin Johnson, and Li Fei-Fei. Hidden: Hiding data with deep networks. In *Proceedings of the European conference on computer vision (ECCV)*, pages 657–672, 2018.