VILNIUS UNIVERSITY

FACULTY OF MATHEMATICS AND INFORMATICS

COMPUTER MODELING STUDY PROGRAMME

Scientific Research Paper

Watermark-Based Viewer Identification in Live Streams

Žiūrovo identifikacija transliacijose naudojant vandens ženklus

Deivis Zolba

Supervisor : prof. dr. Igoris Belovas

Scientific advisor :

Reviewer :

Vilnius
2026

Summary

The study aims to introduce a scalable watermarking model that could be used to design a fast watermarking tool that is invisible and robust enough to identify the content source–viewer. This tool would enable adequate digital copyright protection on a large scale. To this end, a survey of watermarking methods that could embed unique viewer information in real-time has been carried out. The model can reliably scale frequency-domain watermarking techniques for mass application.

Keywords: digital watermarks; data protection; user identification; copyright enforcement; combinatorial optimization.

Santrauka

Nagrinėjami metodai, kaip integruoti žiūrovo identifikavimo informaciją į skaitmeninį turinį, tokį, kaip vaizdo įrašai ar nuotraukos. Dažnių srities vandens ženklinimo modeliai pasižymi dideliu atsparumu suspaudimui, regionų iškirpimui ir kitoms atakoms, tačiau jų taikymas masiniu mastu susiduria su reikšmingais skaičiavimo kaštais. Įprastas taikymas reikalauja itin daug skaičiavimo išteklių, todėl nėra praktiškai pritaikomas plataus masto naudojimui.

Siūlomas sprendimas suskaido dažnio sritį į segmentus ir sukuria ženklų komponentus, kuriuos galima įvairiai kombinuoti. Šis modelis yra gerokai efektyvesnis ir lengviau plečiamas: jis sumažina skaičiavimo sąnaudas ir leidžia sugeneruoti daug unikalių vandens ženklų variantų iš riboto komponentų rinkinio. Sprendimas plečiasi logaritmiškai augant N žiūrovų kiekiui, daug pranašiau už įprasta tiesini augimą.

Raktiniai žodžiai: skaitmeniniai vandens ženklai; duomenų apsauga; žiūrovo atpažinimas; autorių teisių apsauga; kombinatorinė optimizacija.

Contents

1 Introduction

Digital watermarking is a branch of information security focusing on embedding and concealing information within digital media to ensure data protection. Such watermarks have multiple goals, including copyright protection, content authenticity, and temperament detection. Classical watermarks are still widely used, but differ from their digital counterparts in that they are visible to the naked eye. In contrast, digital watermarks are invisible to human eye observations.

The field of digital watermarking is rapidly evolving in response to the growing need to protect intellectual property. The COVID-19 quarantine period saw a dramatic surge in digital media consumption, driven by increased use of entertainment, communication, and remote work platforms. This rise in online activity led to a significant increase in the creation and sharing of digital content, making digital watermarking more essential than ever for safeguarding intellectual property and preventing unauthorized distribution.

1.1 Problem statement

Digital watermarking technique is widely used to protect digital media. However, most traditional watermarking techniques embed a single watermark into the content, which remains the same across all media copies. Such an approach cannot identify the origin of the data leak, rendering it impossible to detect with forensic analysis.

This research addresses this limitation, proposing a watermarking system that embeds unique, user-specific watermarks into media. Each viewer would perceive that they watch the same video as everyone else, but it would be invisibly unique and watermarked with user-specific details. Having that achieved system would allow precise identification of the leaker.

1.2 Problem relevance

The growth of digital media consumption has been steadily increasing and skyrocketed during the COVID-19 pandemic. Digital media providers must continually enforce copyright protection to combat piracy and prevent unauthorized sharing. While efficient tools already exist for removing copyrighted material from the internet, they only address the symptoms of the problem, not its root cause. This research addresses the issue by providing an efficient solution that should identify the leaker and hold them accountable for violating digital media laws.

Digital watermarking offers an effective solution to this problem by embedding unique marks into the content viewed by each user, enabling precise source identification. However, this approach typically requires creating redundant copies of the original media, which poses significant challenges for large-scale deployment. Additionally, making such copies takes time and can not be realistically done during a live broadcast. The need to generate an individual copy for every user is highly impractical in terms of storage space. Therefore, it is crucial to investigate alternative methods to maintain the same functionality while overcoming limitations.

1.3 Aim

This research aims to develop an information security tool using a watermarking technique to embed unique viewer information during live broadcasts.

1.4 Research objectives

1. Overview, analyze literature, and explore digital watermarking research. Find out what the drawbacks are, what could be improved, or what needs further study. Find relevant experiments and results that can be used to compare my findings.

2. Raise and analyze the requirements (see Section 3) for a real-time video watermarking system; such a watermarking technique should efficiently keep viewer information.

3. Propose a scalable and working real-time watermarking model. Define the pipeline for a memory-efficient application for graphics card processing.

4. Implement and test the designed watermarking model. Evaluate the speed and practicality of the model.

1.5 Methodology

The system is developed following a modular methodology that combines signal processing, cryptography, and eventually real-time video streaming. First, the video stream is divided into frames or segments. Next, they are transformed to the frequency domain (e.g., using FFT) to apply the watermarks. Parallel processing with GPU acceleration (CUDA) enables efficient handling. By following the proposed model, computational and storage needs are sufficiently improved to yield promising results for mass live broadcast applications.

1.6 Novelty and practical value

This research introduces a framework for mass-application, per-viewer watermarking in real-time video streams, an area that currently lacks significant research in the literature. Unlike the traditional watermarking approach, which does not scale for mass real-time applications, the proposed model enables the individualized, real-time embedding of unique watermarks for each viewer, making leaks traceable.

The system provides practical applications for digital media security and rights management. Enabling streaming broadcasters to protect content by embedding imperceptible but retrievable watermarks into each distributed broadcast. This solution could be used to fight piracy or protect sensitive video streams, introducing accountability in digital media consumption.

Theoretical work that leads to proving Lemmas and Theorems to improve proposed model.

1.7 Approbation

Preliminary results of the research has been presented at the national scientific conference:

- Efektyvus vandens ženklų taikymas dideliū žiūrovų skaičiui identifikacijai. Lietuvos matematikų draugijos LVXI Konferencija, Klaipėda, 26-27 June 2025.

Preliminary results of the research has been published in a peer-reviewed scientific journal:

- Zolba, D. (2025). Efektyvus vandens ženklų taikymas dideliam žiūrovų skaičiui identifikuoti, Lietuvos matematikos rinkinys, 66(B), p.139–150. DOI: 10.15388/LMR.2025.44464. [16]

1.8 Structure of the thesis

The structure of the thesis is as follows. The Introduction (Section 1) outlines the motivation and background of the work. Section 2 provides the theoretical foundation by analyzing prior research in digital watermarking. In Section 3, the requirements for a real-time viewer identification system are defined. The same chapter also examines the theoretical basis of the proposed model, including its mathematical optimization, stage complexity analysis, all of this leads to the practical implementation, efficiency evaluation, visual impact analysis, and assessment of the model's viability.

2 Digital watermarking: theoretical background

2.1 State-of-the-art in digital watermarking: a literature review

Early research primarily explored images for copyright protection and copy control [9]. Through the years, the watermarking technique has expanded to other media, such as audio and video [9] and even text documents [13]. This section reviews the most recent principal studies on digital watermarking, focusing on watermarking domains, method classifications, and the primary objectives of digital watermarking.

Chen et al. (2023)

Real-time robust watermarking was recently explored by Chen et al. [3]. Digital watermarking, unlike standard steganography, embeds traceable signals into digital content. The practical aspect of this study was done on a Windows desktop rendering pipeline, meaning it's tied to the viewers' environment and not done on digital media itself.

The research explored one of the most challenging problems of digital watermarking - screen-camera, recording the screen with an external camera. Creating a copy of the original media in such a way introduces considerable distortions, and the values of individual pixels get degraded unrecognizably, rendering most digital watermarking techniques ineffective. Another problem is the perspective change since recording often happens in an angle [3]. The only effective watermark technique that survives such an attack is visible on-screen watermarks such as logos and text. Such elements degrade viewers' experience and are easily removed while being obvious. These problems created a clear need for an invisible watermarking system that is robust enough to capture cross-media.

One way to address the perspective problem is to use reference points to ”flatten” the image. Luckily, these points can be easily selected to be the corners of the screen. Automatically embedding special fiducial patterns or using deep neural networks to withstand perspective changes, Moire patterns, and noise is also possible. While effective in controller settings, these techniques tend to be computationally expensive [3].

Chen et al. design a resolution-adaptive watermark template combining an encoded message pattern with two synchronization mark patterns. The watermark message - containing identifying information such as user ID and current time is first encoded as a QR code, chosen for its error correction capabilities and resilience [3]. This message is transformed into the frequency-domain template using a discrete Fourier transform and adjusted with special synchronization signals. The synchronization patterns play an important role as reference markers that help locate and correct watermarks. For embedding, the authors took a system-level approach: working directly with Windows Desktop Window Manager, which is responsible for rendering the GUI. Inserting a watermark proved highly efficient since they can process each display frame before sending it to the display. Their implementation blends an invisible watermark into each frame in real-time by intercepting the swap chain, a mechanism that swaps rendered frames. To ensure the watermark remains imperceptible to the viewers' eye, they use a simplified Just Noticeable Difference (JND) model, adjusting water-

mark strength according to the currently displayed screen content [3]. This adaptive approach means bright or textured areas might carry a stronger watermark signal, and smooth areas weaker ones to avoid visible artifacts. The rendering level approach is the key originality of this work. Instead of modifying content requiring more computational power, the watermark embedding happens within the operating system rendering pipeline with simple addition operations, making it incredibly fast. The result is incredible; any content shown on the screen - images, video, multiple apps, or documents are already watermarked in real time.

In summary, Chen et al. introduced an innovative approach of having the watermarking process done on the operating system level. This gave a fast, true real-time watermarking solution that is even resilient to external camera recordings. Working on the OS level made the technique work on any screen content since the watermark is applied at the display level. While work is transformative, it has many limitations. One of them is platform dependency, the current approach is tied to Windows DWM framework. Another one is software screenshots, screen recordings would not carry this watermark, since it only happens on user display, it only works for external camera recordings.

Xiao Yang and Zhenzhen Zhang (2023)

The authors proposed a new robust watermarking algorithm designed to improve 2D-DFT (two-dimensional discrete Fourier transformation) [7]. The primary motivation was to enhance embedding capacity since the original algorithm had an unpractical capacity of 1 bit per group of frames. While DFT-based video watermarking offers excellent robustness, it tends to be vulnerable to attacks such as frame-dropping. The authors presented improvements to enhance the watermark embedding technique to tackle these limitations.

The main idea was to construct a new DFT template that allows multiple bits to be embedded in each Group of Pictures (GOP); doing so would improve watermark carrying capacity. By employing a prime factorization method for template design, the algorithm could theoretically embed n bits per GOP instead of the previous 1-bit limit [7]. Additionally, adjustments to the template design improved resistance to frame deletion attacks by segmenting frames. In practice, rearranging or adjusting the GOP pattern so that if frames are dropped or skipped, the watermark can still be recovered from the remaining frames. This paper improved watermark capacity and robustness simultaneously, ensuring the watermark survives possible attacks. The authors also compared their algorithm to other state-of-the-art video watermarking techniques. The analysis shows that the proposed method archives competitive or superior performance in multiple aspects - it has great robustness, offers real-time efficiency suitable for practical use, and significantly increased carrying capacity [7]. This study explores and addresses conflicting goals of capacity vs. robustness.

Despite its success, the proposed technique still has limitations, specifically in handling composite attacks. The robustness evaluation in the paper considered various attack types such as frame dropping, cropping, noise, etc., mostly in insolation. However, real attacks could involve combinations of transformations; video might be cropped, rotated, and scaled at the same time. The technique's ability to resist multiple simultaneous distortions is not explored or discussed. Another point of possible analysis is in increased capacity; though the paper presents theoretically n bits pet GOP,

clearly, there exists a practical upper bound for n in a realistic implementation. Additionally, tremendous values of n would require significant modifications per frame, potentially making the watermark visible.

Guoyuan Lin and Weiqi Luo (2025)

The study introduced a novel solution called the DeepAWR deep-learning-based audio watermarking method against AR distortions [8]. The re-recording attacks are very challenging since recording devices don't capture the original signal perfectly. Having original recording transmitted through speakers and a copy being made by a microphone results in a unique copy of the original media, often destroying or severely degrading it and any embedded watermark [8]. Traditional audio watermarking techniques are robust against common signal distortions such as compression and noise addition, but they struggle with complex adjustments such as re-recording. The main idea of DeepAWR is to include deep learning to make audio watermarks resilient to re-recording attacks. The authors designed a deep neural network-based framework for embedding and extraction processes. Each watermark bit is embedded sequentially into low-frequency components of the audio's magnitude spectrogram, frame by frame [8]. The focus is primarily on low-frequency ranges, as they are both more robust and less perceptible. Low frequencies are more likely to withstand playback and recording processes, whereas high frequencies are more susceptible to distortion from air and device limitations. Moreover, the human ear is less sensitive to subtle alterations in lower-frequency sounds.

An et al. (2024)

The authors introduced a multipurpose video watermarking algorithm that combined tamper detection with copyright protection [1]. The method incorporates two types of watermarks: a DFT template - resistant to rotation and scaling and a copyright mark DCT. The DFT template allows geometric correction to be done during extraction, which ensures watermark robustness against transformations [1]. Meanwhile, the block-wise embedding of the DCT watermark has validation bits, and they allow localization of temperaments. Results show that the technique watermark survives - compression, scaling, and rotation proving the algorithm's suitability for both robust protection and forensic tracking [1].

Hou (2021)

Hou presented an MPEG and DA-AD resilient DCT-based video watermarking scheme using adaptive frame selection [6]. The watermark is hidden within the spread spectrum, as shown in Figure 2. It is embedded into the DCT coefficients of selected frames, where the frames are chosen adaptively to maximize resistance to desynchronization. This method has been proven to survive and remain retrievable after MPEG-4 compression [6]. To make the watermark resistant, the authors repeat it multiple times, increasing the resiliency against various attacks. The design allows the extraction of a 16-bit watermark from any 15-second video interval, making it suitable for long videos.

The implementation meets IHC's robustness criteria and is lightweight enough for integration into real-time encoding systems [6].

Sun and Srivastava (2023)

The authors proposed a digital watermarking algorithm based on locality-sensitive hashing (LSH) and wavelet transform [14]. The video is decomposed using a one-dimensional wavelet transform, after which the watermark is embedded in both low and high-frequency bands using hash-based values and XOR operations. This dual structure targets both authentication and robustness against various attacks. Furthermore, singular value decomposition dynamically regulates the strength of watermarks. The results demonstrate precise tamper localization and high PSNR values (approximately 49 dB), indicating outstanding visual quality and resilience to multiple attacks, including JPEG compression and noise [14].

El-Shafai et al. (2024)

Previously explored techniques hide information in the transform domain; however, there are more primitive techniques that embed a watermark directly into the spatial domain. LSB substitution embeds the watermark, changing the least significant bits of pixel values in the spatial domain [12]. This technique is straightforward and undetectable; the host image or video looks the same, yet the embedded watermark is highly vulnerable and fails against nearly all attacks [12]. Typical attacks, such as lossy compression, noise addition, or filtering, can easily destroy an LSB watermark; therefore, LSB is mainly used where any alteration should nullify the watermark, e.g., fragile authentication marks [9]. The advantage of LSB-based schemes is their minimal computational cost, which makes them perfect for real-time embedding [9]. Creating a unique LSB watermark for each viewer is simple in live streams, but its lack of robustness renders it unsuitable for source-tracking applications.

ETSI (2023)

Researchers at the European Telecommunications Standards Institute (ETSI) proposed the way to identify a viewer by A/B watermarking [5]. A/B watermarking creates two video versions: variant A encodes a ”0”, while variant B encodes a ”1”. Each viewer is assigned a unique bit pattern, which, at playback, is represented by either A or B segments. This results in a viewer-unique sequence without significantly altering the pipeline. This pattern is spread across time by having multiple second segments, where each segment contains all frames of either type A or type B. In watermark extraction, frames are then identified by comparing them to see if they are more like A or B. This redundancy provides adequate robustness, but results in extraction requiring lengthy copy - several minutes for identification [5].

2.2 Principal watermarking domains

2.2.1 Images

Digital watermarking in images generally conceals information within the pixel data of the image. Image watermarks can be visible, such as a transparent logo or text overlay, or invisible. Visible watermarks, such as a company logo, clearly indicate ownership but can be intrusive. In contrast, invisible watermarking embeds data imperceptibly to the human eye, yet the hidden data can be recovered with the correct technique.

Two main strategies exist for invisible image watermarking: spatial and transform domain methods. Spatial domain techniques embed the watermark directly into selected pixels. The simplest example of this is the least significant bit (LSB), where the lowest bits of selected pixels are altered to encode the watermark. Being straightforward, undetectable, and requiring minimal computational needs does not come without downsides; it cannot survive even the simplest geometric attacks [9]. By contrast, transform domain methods first convert the image into the transform space, and then embed watermarks into the transform coefficients. Ways to transform data into the transform domain include the Discrete Cosine Transform (DCT), the Discrete Wavelet Transform (DWT), and the Singular Value Decomposition (SVD). While the transformations are costly, embedding information in only specific frequency ranges makes the changes very sudden and unnoticeable to the human eye. To better understand how these techniques work, we can imagine the transform domain as a pattern of values representing our image, and changes made to that pattern will affect all other image pixels when we convert back to the spatial domain [1]. In that sense, our embedded data is not hidden in one area or specific pixels, but rather throughout the image as a whole. While this is a significant oversimplification of the concept, understanding that we can explore why these methods are more robust and resilient to standard geometric operations, as well as compression. While LSB cannot survive cropping, resizing, or other basic geometric transforms since embedded data is exceptionally fragile, transform domain watermarking spreads data across all pixels of the image; for this reason, they tend to be more robust to such operations. Although scaling or cropping part of the image may damage the watermark, it can often still be recovered [9]. Under compression, while the absolute pixel values change, the transform domain structure in which our watermark is hidden is mostly preserved.

2.2.2 Video

Video watermarking extends the concepts of image watermarking to a sequence of images (frames). Many techniques apply image watermarking algorithms to each frame. However, the video domain introduces new challenges. The biggest challenge is real-time processing; videos have high frame rates, so if watermarking is done in real time during live broadcasting or streaming, the embedding process must keep pace with the video playback speed. For this reason, simple visible watermarks, such as a broadcaster's logo, are displayed in the corner of the stream.

Additionally, video watermarking methods often fall into two categories: frame domain and compression domain. Frame domain techniques embed the watermark directly in the video's frames,

utilizing any of the image-based watermarking methods. This is exactly as it sounds: each frame is just an image across the timeline of the video. Some schemes could even expand the idea of carrying a watermark through a group of frames. Compression domain techniques, on the other hand, embed the watermark during the video encoding process. For example, as an MPEG or H.264 video is encoded, the algorithm inserts an additional step that tweaks specific DCT coefficients [7]. The advantage of embedding while literal compression occurs is efficiency - the watermark is inserted in real time with minimal extra processing costs. It often integrates nicely with existing codec operations for broadcasting purposes. However, a watermark is tied to a specific compression scheme, and it may lose robustness if the video is transcoded (i.e., a change of format or codec).

For personalized watermarking, robustness is the most crucial consideration; a good watermark should withstand common video manipulations, such as re-compression, resizing, or changes to frame rate. While re-compression does not sound ”common”, it really is; each re-upload of leaked content, even unintentionally, will probably be re-compressed. Let us imagine a screenshot or screen recording was taken from our broadcast and uploaded to a forum, social media website, or any other platform. The website would process and normalize it to its standard, saving on storage and bandwidth costs. This will most likely involve compression and degradation of the content, or even a format change.

2.3 Watermarking method goals

Digital watermarking involves embedding hidden information into a host signal (image, video, etc.) for various purposes. While designing a digital watermark, knowing its goal is crucial, as each purpose has unique needs; some marks are designed to disappear under any transformation, while others are intended to withstand compression or cropping attacks. At a high level, I have clustered objectives into a few groups, reflecting how the mark will be used in practice.

2.3.1 Copyright protection

Digital watermarking for proof of ownership is probably the most widely used type of digital watermarking. There are already numerous viable and efficient solutions to achieve this goal. Watermark typically encodes the owner’s ID or logo and is usually kept imperceptible to avoid affecting the user’s experience. In some cases, a visible watermark or a translucent logo is used, but not for the reasons we might assume. While it can prove ownership, it serves primarily as a marketing tool used to advertise the origin of this content indirectly.

The uniqueness of this goal lies in carrying static identifiers that identify the owner’s information and remain the same in all copies of the content. The watermarks often need to hold up in court or formal disputes, so they should be robust enough that their detection yields high confidence. While proof of ownership is important, once the rightful owner finds a copy of their media, numerous tools are available to request a takedown. For example, in the EU Digital Services Act (DSA) or the EU Copyright Directive (DSM), even GDPR Article 17 (”Right to be forgotten”) covers the takedown of personal data. Google, as well as other search engines, follow the EU privacy removal process.

They also comply and will remove the index of a violating website even if the website is from a non-compliant region. Watermarks can simplify the processes, but even without them, copyright can be enforced.

For bigger players in the industry, digital watermarks or even fingerprints of content can help automatically detect copies. Most commonly known for music indexing, the idea works similarly to the Shazam app, which can identify music from a few seconds of a song. YouTube does the same for videos, automatically assigning the rightful owners and removing any monetization that was made from unlawful copies. But this is less related to the digital watermarking we are discussing.

2.3.2 Content authenticity

When the goal is to ensure content has not been tampered with, fragile or semi-fragile watermarks are typically used. A fragile watermark is designed to be destroyed or visibly damaged if any changes are made to the host content. If the content is modified, the extracted watermark no longer matches, showing tampering. Spatial domain techniques are ideal for this, as they directly associate the watermark with the pixel or bit values of the content. In some cases, a semi-fragile approach is used - the watermark tolerates certain unintentional changes (e.g., JPEG compression) but breaks under malicious changes. This is achieved by encoding content features inside the watermark. Watermark designs for authenticity often also aim to localize where changes occurred. This technology is widely applied in CCTV cameras, sensitive communications, medical records, archives, evidence validation, and forensic analysis.

The central distinction regarding this goal is sensitivity vs robustness; the watermark must be sensitive enough to detect unauthorized alterations, yet not so sensitive to trivial or acceptable changes. Semi-fragile techniques adjust for this by defining which transformations are allowable. Designers sometimes combine cryptographic signatures with watermarking for double protection. These schemes utilize secure keys or one-way functions in embedding. This is used to prevent possible forgeries and replications of authenticity seals. To conclude, content-authentication watermarks prioritize integrity verification; they act as a fragile seal on the media. Their designs often work in conjunction with checksums or hashes, and are uniquely focused on being easily breakable by changes, yet trustworthy under normal data handling.

2.3.3 Forensics and Tracking

Forensic and tracking watermarks are unique in that they tie a piece of media to a specific source or recipient. A common technique is digital fingerprinting, where each copy of the content is assigned a slightly different watermark that carries personalized data, such as a unique identifier. Technically, these are just robust watermarks with the added requirement that each copy carries personalized data. Due to the possible similarity of watermarks between copies, there is an additional need for error correction and redundancy. This is crucial to avoid possible false positives and misidentifications if watermarks degrade, which could lead to incorrect identification. In video or audio, the watermark can be spread across time, increasing payload and robustness. Spreading watermark through time frame is the core idea of A/B watermarking [5].

Having multiple copies of the same recording can introduce additional challenges, such as collusion attacks. If multiple users each have watermarked copies, they can compare their copies to identify any differences. This can lead to successful attempts to remove or forge watermarks. Therefore, for this goal, watermark designs must additionally have some collusion resistance. Another challenge is maintaining anonymity and privacy: the watermark might identify a user, but it should not be accessible to anyone. Designs must ensure that only authorized parties can interpret the watermark, while outsiders cannot read or fake it. Additionally, illegal distributors will do everything to destroy any tracking marks, making robustness vital. For all these and other reasons, system complexity increases, one must manage a database of watermark mappings, secure key distributions, and adaptive algorithms to obfuscate the designs further.

Main practical application areas where forensics & tracking watermarks are used:

- Film & TV distribution: for example, unique per-copy watermarks are embedded in theatrical releases to trace leaks back to the theater. Currently, this is less common, but previously, many recordings of theater screens were done with an external camera. Here, watermarks indicated a specific theater and possibly the time of recording, helping identify the purportrator [12].
- E-books & digital publishing: Each purchased copy can be watermarked with a unique identifier, allowing redistribution versions to be traced back to the original buyer.
- Document sharing: forensic watermarks in PDFs or scanned documents, tracing of leaks of insider threats. The most notorious incident involved the tracking dots on Xerox and Canon printers. This is a physical watermarking technique, where nearly invisible yellow dots were placed on all printed content by the printer. These dots were able to identify the printer by its serial number for over 30 years until it was leaked to the public.
- OTT, Streaming platforms: Session-based watermarks, such as A/B watermarking, are employed, allowing operators to identify which account or device restreamed and pirated the content. This application is the main goal of this thesis. While A/B watermarking is effective, it has certain limitations that my proposed model aims to overcome.

2.4 Watermarking methods classification

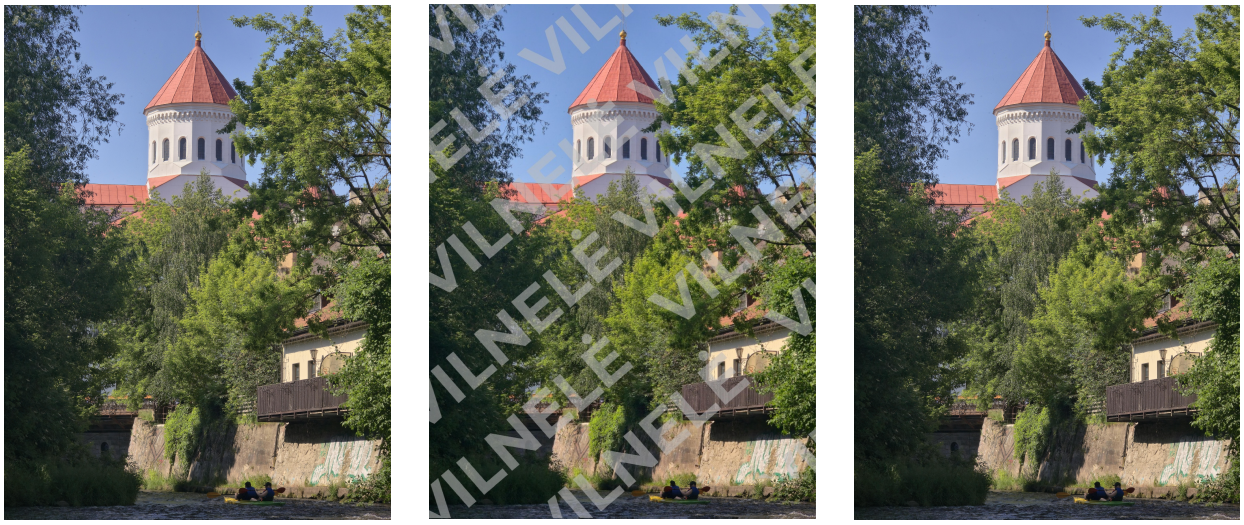
When designing a watermark, numerous aspects must be considered, and a single watermark can not do it all, as some aspects are inherently conflicting. It's possible to layer multiple different techniques, but they can start to cannibalize each other. When designing watermarks, it's essential to consider the critical characteristics for the application.

2.4.1 Visibility: Visible vs. Invisible

Visible watermarking embeds information in a way that makes it perceptible in the media content, such as a logo [13]. The watermark is intentionally observable by viewers, which tends to deter unauthorized use by clearly indicating ownership. Because it's so easily identifiable, it can be easily altered or removed by an attacker.

In contrast, invisible watermarks are, as the name implies, invisible. By design, they carry embedded data in a way that cannot be easily observed by simply viewing or analyzing the data [8]. This approach is widely used for copyright protection, as it does not impact the viewer's experience while providing a means to confirm ownership. For example, in an audio watermarking study, choosing a high signal-to-noise (SNR) ratio in the range of 34-36 dB means the embedded audio watermark was nearly inaudible to listeners [8]; this is analogous to being invisible in images, being imperceptible to the eye. To further illustrate the difference see Figure 1.

Both types have their use cases: visible watermarks declare their presence upright, whereas invisible ones give the same guarantees without alerting the viewer about the watermark's presence.



a) Original b) Visible, text - "VILNELĖ" c) Invisible, text - "Vilnelė"

Figure 1. *Watermark visibility types, Visible vs Invisible*

2.4.2 Robustness: Robust vs. Fragile

Watermarking techniques are also distinguished by how they react to modifications in host content. Robust watermarking is designed to withstand common signal processing attacks, ensuring the watermark remains retrievable even after compression, resizing, noise addition, and other similar operations. This property is crucial for tracking goals, as the perpetrator is expected to intentionally try to remove or damage the watermark.

Fragile watermarking is the complete opposite; it's crafted to break or disappear under minor editing [13]. Such watermarks serve as an authenticity property; any change to the content destroys the watermark, later giving confirmation that the original content was somewhat modified or tampered with by an attacker [11]. A fragile watermark can be thought of as a tamper-evident seal. When your water bottle seal is intact, you can be confident that it is original. Once opened, it cannot be restored, so a broken seal immediately shows that tampering has occurred.

In summary, robust watermarking is suited to survive attacks, while fragile ones should indicate that the media has been modified and is not authentic.

2.4.3 Detectability: Private vs Public

In watermarking, detectability refers to the criteria needed to determine whether a watermark is present. Private or non-blind schemes require additional side information, typically the original media before the watermark was applied, or a secret key that determines pattern zones or decryption. Public or blind schemes don't require the original media or any additional information; watermark creation might need a private key, but extraction typically does not. This is designed in such a way that anyone can verify the presence of a watermark, but only the owner can embed it [9].

These differences create clear trade-offs. Private detection benefits from stronger side information, which in turn typically yields lower false-positive rates for improved robustness. Blind detection must work without the original and relies on thresholding. Designers must also distinguish between achieving detection (i.e., determining if a watermark is present) and extraction (i.e., recovering the watermark) as separate goals due to the need for robustness and potential degradation. For this reason, A/B watermarking compares a group of frames more closely to version A or B, carrying only a single bit of information across a group of pictures, as recovery is not reliable; hence, determining the presence is all that is needed, and this is a reasonable approach.

2.4.4 Reversibility: Reversible vs. Irreversible

Most digital watermarks change the original signal permanently. Reversible watermarking, also called lossless or invertible watermarking, lets original content be restored after watermark extraction[9]. Reversible techniques are used where original data is crucial, such as medical images and military and legal records[9]. For example, a reversible watermarking method for images might embed a watermark into the image's pixel values or coefficients, allowing for the retrieval of the original image losslessly and the hidden message during extraction. Sigh and Kasana (2024) note that such methods ensure the integrity and authenticity of documents[13]. The drawbacks of reversible watermarking are that it typically carries a smaller payload, is less robust, and tends to be more fragile since it cannot make irreversible modifications.

Irreversible watermarking is a more traditional approach, where the watermark permanently and irreversibly changes the watermarking media. Once embedded, original content cannot be restored without information loss. Irreversible methods happen naturally, where slight distortions of original content are acceptable and not taken into account. Not having to worry about making the technique reversible gives big freedom. With fewer limitations, irreversible techniques can carry more information and are more robust and durable.

Taking everything into account, reversible watermarking is used where the original content must be preserved [13], whereas irreversible watermarking can embed more deeply and change media permanently; this enables higher payload capabilities while being more durable.

2.5 Possible approaches, to achieve viewer identification

2.5.1 Metadata embedding

The simplest and most cost-effective way to achieve this would be to carry that information in metadata. However, metadata can be easily changed or even destroyed. For example, if a user creates a copy while screen-recording the displayed media, metadata is not captured in most cases.

2.5.2 Visible watermarking

Another way would be to add a visible watermark to the media content itself. This method makes it immediately apparent who the intended recipient is; this could deter some unauthorized sharing. Unlike metadata, visible watermarks would survive and be captured even during screen recordings and screenshots, making them more robust against casual piracy. However, this approach will introduce some friction and degrade the viewing experience. Watermarks could be distracting, break immersion, and obstruct parts of the content. Since the watermark is visible, skilled attackers could crop, blur, or destroy it, making it inefficient against basic attacks.

2.5.3 Invisible watermarking

The most promising way in terms of survivability and robustness to even ”accidental” attacks, such as compression, is frequency domain watermarking. Embedding a watermark inside the frequency domain would spread it throughout the entire media, and part of it would exist in every pixel. This is why such a technique remains effective even after scaling, cropping, and compression. All is well until we consider the computational cost of watermarking in the frequency domain. While relatively inexpensive, the costs scale horribly for mass applications.

Videos typically have a frame rate in the range of 25-60; the pixel amount varies depending on the quality of the media. Popular resolutions are listed in Table 1.

Table 1. *Video resolutions and their pixel counts.*

Resolution	Dimensions	P (Pixels)
HD (720p)	1280 × 720	921 600
FHD (1080p)	1920 × 1080	2 073 600
QHD (1440p)	2560 × 1440	3 686 400
UHD / 4K	3840 × 2160	8 294 400
8K	7680 × 4320	33 177 600

Let us consider the Fast Fourier Transform (FFT) for watermarking. Firstly, we convert our video frame to the frequency domain, which has a complexity of O(P log(P)), where P is the number of pixels in the frame. We need to insert the watermark, which has O(W) time complexity where W $\leq$P, and after all that, we need to convert the signal back to the spatial domain, doing inverse conversion O(P log(P)). The flow can be further seen in Figure 2.

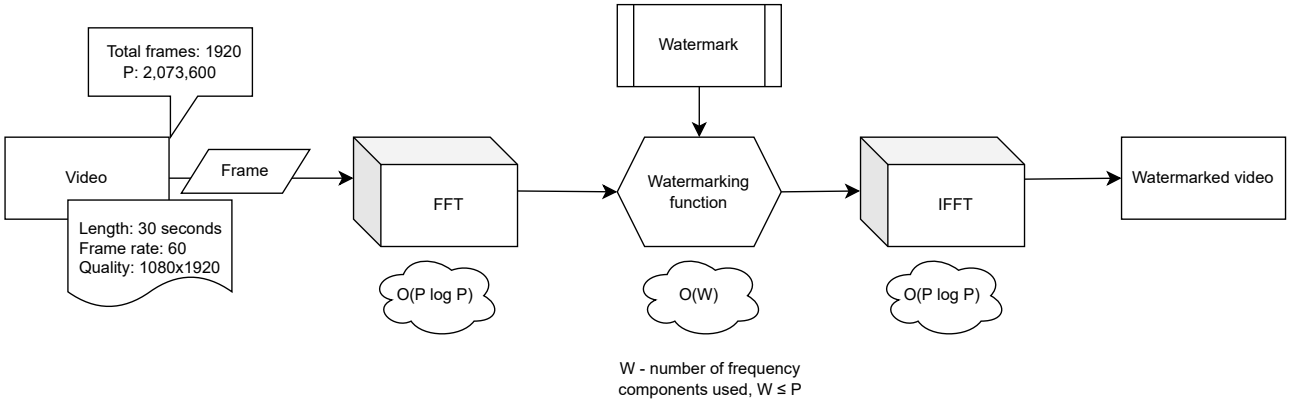


Figure 2. *Watermarking process using FFT and IFFT on a 1080p video frame.*

The simplest way to uniquely identify the viewer would be to use different watermarks in the watermarking step and require an inverse FFT for each variant. We can save on the forward FFT since it can be copied and reused by each applied watermark. Even if the watermarking part was completely free with some negligible cost, there is another elephant in the room: encoding.

After applying watermarking algorithms, we are left with raw data, which is not renderable. Video encoding is a process that converts raw video data into efficient, standardized bitstreams. This transformation is essential to ensure compatibility with internet delivery protocols while optimizing bandwidth usage. Modern codecs (e.g., H.264/AVC, H.265/HEVC, AV1) remove spatial and temporal redundancy via block transforms, basically optimizing and arranging raw data. Encoders also introduce the Group of Pictures (GOP) structure, introducing several new concepts, including I-frames, P-frames, and B-frames. For adaptive streaming, the same content is encoded into an adaptive bit rate (ABR) ladder, allowing clients to switch resolutions and bitrates based on varying throughput conditions. Also, there are standardized pixel formats and color spaces, among other specifications. Collectively, encoding enables high-quality, efficient transmission and interpretable content across various devices and networks.

Since the encoding standardizes our data, the watermark becomes degraded indirectly. Specific watermarking techniques, as previously discussed, leverage the encoding phase to their advantage by embedding the watermark during the encoding process. This links the watermark to its encoding, resulting in decreased robustness against re-encoding or even capture through screen recording. Encoding-based watermark techniques don’t fulfill the requirements introduced in Section 3.

3 Real-time digital watermarking: requirements, challenges, and model proposal

The primary objective is to develop a comprehensive solution for video broadcasting that enables the detection of leaks without compromising the user experience or interrupting content delivery. This solution also requires a real-time, unobtrusive, and robust digital watermarking model that assigns a unique identifier to each viewer for every stream. This identifier must remain detectable even after common processing tasks, such as compression, scaling, cropping, and screen recording.

Design should reliably carry a payload, a viewer-ID, that can be recovered from short clips. Scalability is the most significant challenge; the system should be economical enough to support thousands of concurrent viewers.

3.1 System requirements

The following lists summarize the architecture requirements. The functional requirements capture what the watermark must communicate. Non-functional requirements display how the system must behave in production. These constraints are meant to guide choices.

Core functional requirements

- **Payload:** The watermark must carry enough information to identify the viewer.
- **Robustness to transformations:** The watermark should tolerate mild compression, format changes, scaling, cropping, frame manipulations, drop/insert/swap.
- **Recoverability:** The watermark should be recoverable from a short recording, even under adverse conditions, with only one frame.

Non-functional requirements

- **Scalability:** The system should scale reasonably (no worse than linear time complexity) as the number of viewers increases.
- **Codec and format:** The model should remain effective across H.264/AVC, HEVC, and other state-of-the-art codecs. It should be versatile and generalized, without being tied to a specific codec.
- **Real-time:** The system should be capable of watermarking in real-time.

Technical challenges

- **Throughput and scaling:** Per-viewer embeddings can lead to substantial increases in both computational demands and input/output (I/O) bandwidth utilization.
- **Storage:** Storing copies of original content grows quickly, especially if high resolution and quality are preserved. For instance, a 1 GB title should not become 1 TB for 1000 viewers. To avoid this in practice, either the personalized version is created in real time or some scaling, such as A/B session watermarking, is applied. Even

if we store truly (fully, uniquely watermarked) personalized versions, transmission and bandwidth needs would still be impractical.

- **Robustness and imperceptibility:** The watermark should be robust and invisible. Watermarks, being based in the spatial domain, typically are not strong enough to withstand compression. Temporal domain techniques tend to require higher computational costs, in effect being poorly scalable.

3.2 Proposed scalable model

To clarify the concept, let us examine how the frequency domain is associated with the FFT. In image processing, the frequency domain represents an image using sine and cosine functions of varying frequencies, rather than the pixel intensities themselves. Applying a Fast Fourier Transform converts the image from the spatial domain (pixel space) into the frequency domain, showing how rapidly intensity changes across the image. Low frequencies relate to gentle, gradual transitions, whereas high frequencies signify distinct edges and intricate details.

Due to the global nature of the FFT, when embedding a watermark in the frequency domain, the watermark information is spread across the entire image. This is a key advantage: even if only a portion of the watermarked image is available, some of the frequency-modified data and thus the watermark remain detectable. This makes frequency-domain watermarking robust to operations like cropping and compression. This property can also be leveraged for scalability in mass applications.

Using the FFT technique, as previously shown in Figure 2, we can calculate multiple versions of the same frame with different watermarks, resulting in numerous unique versions, as before. Next, let us consider each version as a set of components, rather than a standalone version. We can now combine multiple components to create several unique versions of the watermarks. Moreover, it is crucial to have non-overlapping watermarks. Once mixed, if we return to the frequency domain, both watermarks will be visible, which means they may overlap and become more challenging to recover if they occupy the same frequency ranges.

To generalize the idea, let us formalize the design. First, our working area, the frequency domain obtained after forward FFT, can be split into regions. Each region can carry a watermark, which is referred to as a component of that region. Having multiple components from different regions allows us to combine them, resulting in far more unique variants than just watermarking each copy uniquely. To further optimize unique variant volume, it will be maximized while minimizing total components, achieving an optimal configuration [16].

$R \in \mathbb{N}$, number of regions.

$c_i \in \mathbb{N}$, number of components in region i, for $i = 1, \ldots, R$. Total number of components:

$$\sum_{i=1}^{R} c_i$$

Variant: a choice of one component from each region. Total number of variants:

$$\prod_{i=1}^{R} c_i$$

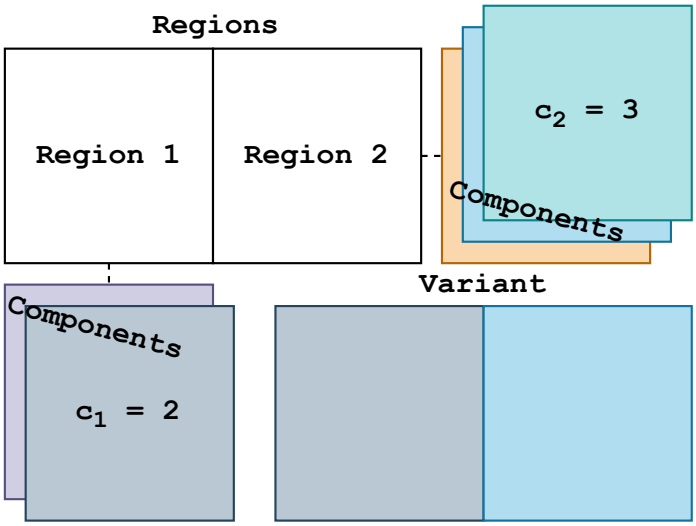


Figure 3. *Proposed model setup explanation.*

Lemma 3.2.1. *AM-GM inequality. Let R be the number of regions, and let c_1,c_2,...,$c_R \in \mathbb{N}$ denote the number of components in each region. By the AM-GM inequality,*

$$\left(\prod_{i=1}^{R} c_i\right)^{1/R} \leqslant \frac{1}{R}\sum_{i=1}^{R} c_i.$$

Equality is achieved if and only if $c_1 = \cdots = c_R$.

Proof. See [2] for several proofs of the AM-GM inequality. Note that the product $\prod c_i$ is maximized when $c_i = C/R$. Here $C = \sum c_i$. Since $c_i \in \mathbb{N}$, the component counts can differ by at most 1, i.e., $\lceil C/R \rceil$ or $\lfloor C/R \rfloor$. □

Theorem 3.2.2. *Let us have each unique variant constructed by taking one component from each region $R \in \mathbb{N}$, and the total number of components in each region is $c_i \in \mathbb{N}$. Then the total possible variant count is*

$$\prod_{i=1}^{R} c_i.$$

If N is total unique variant count, then this equality should hold

$$\prod_{i=1}^{R} c_i \geq N.$$

To reach at least N unique variants requiring the minimum total number of components, Minimum total component amount $\sum_{i=1}^{R} c_i$ is reached when all regions have the same number of components: three,

$$c_1 = c_2 = \cdots = c_g = 3,$$

and the total number of regions will be:

$$R = \lceil \log_3 n \rceil .$$

Proof. Optimization of min $\sum c_i$, when $\prod c_i \geqslant N$, from Theorem 3.2.1 we have min Rc, when $c^R \geqslant N$, where $c = c_i$ for all i. Since $c \geqslant 1$ and $R \geqslant 1$, its enough to minimize the total components $f(R) = Rc^{1/R}$. For $R \geqslant 1$, this function has one minimum at point $R_{min} = \ln n$. So we have,

$$c_{min} = \frac{f_{min}(R_{min})}{R_{min}} = \frac{\ln n \cdot e^{\ln n / \ln n}}{\ln n} = e.$$

If $c,R \in \mathbb{N}$, then $c = 3$ and $R = \lceil \log_3 n \rceil$. Since $c = c_i$, the most optimal way is to have 3 components in each region. □

Remark 3.2.3. In Theorem 3.2.2, if $\{\log_3 N\} < 2 \log_3 2 - 1 \approx 0.261$, we use one fewer region and the last region will have 4 components. Otherwise, we use $R = \lceil \log_3 n \rceil$ regions: the last region has 2 components if the fractional part is less or equal to $\log_3 2 \approx 0.631$; else we keep the last region with 3 components.

Unique variant growth, as components increase, can be analyzed by plotting the graph, as seen in Figure 4. It is a log-scale plot, and the curve rises exponentially, with variants approximately equal to $\approx 3^R$; as each region carries three components. At a cost of computing 27, we have 19683 unique variants, and at 45 components, it reaches more than 14 million. In practice, watermarking has a limit on the number of regions in its spatial domain, as each new region reduces the size of all other regions. For instance, with the 45 components, we would have 45/3 = 15 regions. If our frame is FHD (1080p), each watermark part would be present in 2073600/15 = 138240 pixels.

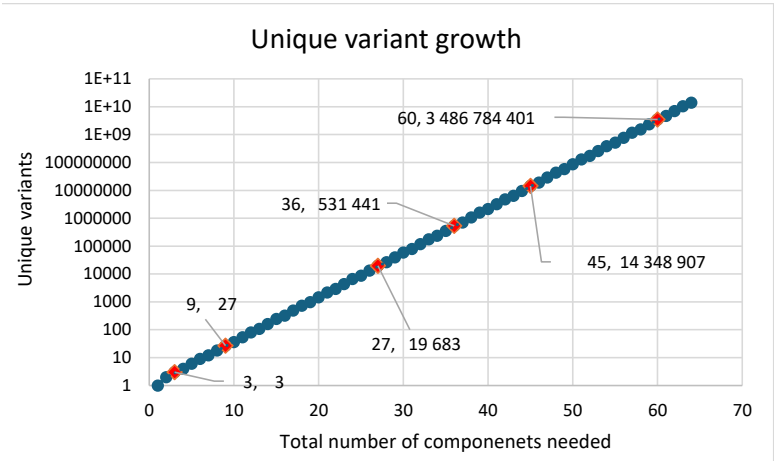


Figure 4. *Variant growth based on Theorem 3.2.2*

This model can be applied in the frequency domain (obtained from FFT) watermarking, effectively reducing the amount of inverse FFT operations needed to perform. The benefit is immense: for instance, instead the need to perform 14 million IFFTs, it is now possible to get away with only 45. Additionally, not only does this save on the watermarking, but also on the encoding.

3.2.1 Storage and Encoding complexity

The proposed model has already reduced storage, encoding, and computational requirements from linear to log-linear. A natural question is whether it is possible to improve it further; can some parts be performed in constant time $O(1)$, independent of audience size?

Storage and encoding are generally the dominant cost factors in media creation pipelines, particularly in systems that require generating multiple media variants. In conventional viewer-specific watermarking approaches, these costs scale linearly with the number of viewers because each version must be encoded separately. This linear dependency is the biggest bottleneck for large-scale distribution systems.

This limitation was addressed by the proposed model by decoupling viewer capacity from the number of encoded assets, introducing a combinatorial variant scheme. Instead of generating a unique media variant per viewer, the system encodes a fixed number of variants per frame, where the number of layers grows logarithmically with the supported audience size. Regarding encoding and storage, their costs no longer scale with the number of viewers but with the number of components. This relationship is formalized by the following encoding and storage complexity lemma.

Lemma 3.2.4. *Let N denote the number of supported viewers. Assume the base-3 unique-variant scheme described in Theorem 3.2.2, where the number of regions is $R = \lceil \log_3 n \rceil$ and each region contains a constant number of components $c_i = 3$. Assume that the watermarking system encodes and stores exactly one media asset per component. Then the total number of encoded and stored media assets is proportional to R, and the encoding and storage system's complexity is $O(\log N)$.*

Proof. By Theorem 3.2.2, supporting at least N viewers is achieved with $R = \lceil \log_3 N \rceil$. As the theorem states, each region has a constant number of components, namely $c_i = 3$ for all $i = 1, \ldots, R$. Hence, the total number of components is

$$C = \sum_{i=1}^{R} c_i = 3R =: \Theta(R).$$

Under the assumption that the system encodes exactly one media asset per component, the total number of encoded and stored assets equals C. Therefore,

$$\text{Encodes} = \Theta(R), \qquad \text{Stored assets} = \Theta(R).$$

Since $R = \Theta(\log N)$, it follows that both encoding and storage complexities scale as

$$\Theta(\log N)$$

with respect to the number of supported viewers N. □

Remark 3.2.5. $O(\cdot)$ is used for asymptotic upper bounds, and $\Theta(\cdot)$ when both upper and lower bounds hold (tight growth). Hence $\Theta(\log N)$ is stronger than $O(\log N)$.

While this is a substantial improvement over the naïve approach, some calculations remain practically discarded. In particular, each component frame is not fully utilized: only a subset of its pixels contributes to constructing a unique variant, leaving the unused pixels redundant. To maintain equal watermark strength and spatial distribution across all variants, only a fraction, $1/R$, of the pixels are used in constructing the final frame. Effectively, only a portion of the pixels is needed; a natural approach is to encode only the pixels that will be used. This can be achieved by partitioning the pixels in the final frame into their respective regions. The improved storage/encoding complexity is analyzed in the following constant time Encoding and storage complexity lemma.

Lemma 3.2.6. *Let N denote the number of supported viewers. Assume the base-3 unique-variant scheme described in Theorem 3.2.2, where the number of regions is $R = \lceil \log_3 n \rceil$ and each region contains a constant number of components $c_i = 3$. Assume that the watermarking system encodes and stores exactly one media asset per component, such that each asset occupies $1/R$ of the original frame. Then the total encoded and stored data per frame is constant, and the encoding and storage complexity with respect to N are $\Theta(1)$.*

Proof. By Theorem 3.2.2, supporting at least N viewers is achieved with

$$R = \lceil \log_3 N \rceil$$

regions, each containing a constant number of components $c_i = 3$. Hence, the total number of components is

$$C = \sum_{i=1}^{R} c_i = 3R.$$

Each component corresponds to a single encoded media asset of size $1/R$ of the original frame. Therefore, the total encoded and stored data per frame is

$$C \cdot \frac{1}{R} = 3R \cdot \frac{1}{R} = 3$$

Thus, the total encoding and storage requirements per frame are bounded from both sides (above and below) by constants independent of N. This achieves $\Theta(1)$ complexity for storage and encoding, with respect to the number of supported viewers N. □

Remark 3.2.7. The constant complexity refers to the total amount of information encoded per video frame. While the number of logical components scales as $O(\log N)$, their individual sizes decrease proportionally, resulting in constant overall encoding and storage cost with respect to the number of viewers.

The complexity analysis established precise asymptotic bounds on the encoding and storage requirements of the proposed model. This demonstrates that as viewers grow, encoding and storage costs do not increase; they remain constant. Separating the growth of logical structure from that of encoded data ensures that the system remains efficient for arbitrarily large audience sizes. Achieved rigorous asymptotic complexity of $\Theta(1)$ for storage and encoding, providing a strong theoretical basis for the practical viability of the proposed model.

3.2.2 Watermarking complexity

This chapter presents a per-frame complexity analysis of the watermarking method. The complexity is defined by two main variables: the number of pixels, P, and the number of supported viewers, N.

Let a single video frame contain $P = W \cdot H$ pixels, where W stands for width and H for height. Typical resolutions were previously discussed, as presented in Table 1. To support N viewers, the proposed model will be used, with the base-3 unique variant configuration. Where the number of regions will be $R = \lceil \log_3 N \rceil$ and each region has three components.

To better understand the analysis presented here, the watermarking process is illustrated in Figure 5.

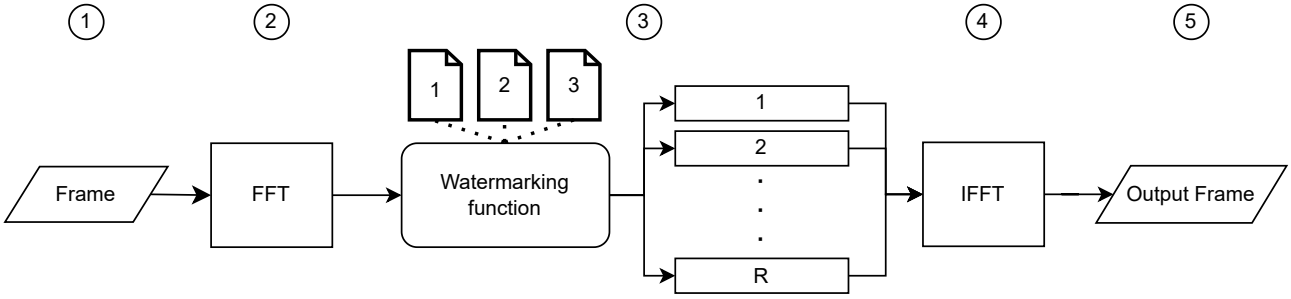


Figure 5. *FFT watermarking.*

1. The process starts with a single video frame, from which the variable P is derived, representing the total number of pixels in the frame.

2. The frame is then passed through a forward 2D FFT, resulting in a frequency domain representation of our frame. The complexity is $O(P \log P)$

3. Watermarking function step modifies a fixed subset of frequency-domain coefficients to embed one of three distinct watermarks. This is the step where the previously discussed components are created. The complexity is $O(W)$, where W is the number of frequency bins modified per component. $W \leqslant P/R$, since watermarks are non overlapping it is applied only on P/R portion of frequency bins per component. In total, there will be $3R$ components, meaning this step is performed $3R$ times, resulting in $3R$ outputs.

4. IFFT is performed to convert the watermarked components back to a pixel- or human-readable format. The complexity is the same as with forward FFT, $O(P \log P)$.

5. Outputted frames are watermarked from previous steps; they are ready to be encoded, stored, or combined for transmission.

Watermark embedding complexity per frame is reflected in the following lemma.

Lemma 3.2.8. *The total watermarking embedding work per source frame is $\Theta(P)$, independent of the number of supported viewers N.*

Proof. The watermarking scheme partitions the frame into R regions, each containing exactly three component variants. For each region, watermark coefficients are embedded independently within the frequency-domain representation.

Since watermarks are non-overlapping and distributed uniformly across regions, where each region spans over - $\frac{P}{R}$ frequency bins. Embedding watermarks in these regions for each of the three components results in a total of

$$3R \cdot \frac{P}{R} = 3P$$

coefficient modifications per source frame; since R cancels out, this step does not scale with the number of viewers N.

Thus, the total watermark embedding cost per frame is linear in the number of pixels $\Theta(P)$ □

The next lemma concerns the number of inverse transforms.

Lemma 3.2.9. *The number of inverse transforms per source frame is $3R = \Theta(\log N)$, not $\Theta(N)$ to support at least N users.*

Proof. Each of the R regions contains exactly three components. These $3R$ components are eventually combined (one component per region) to create up to $3^R \geqslant N$ distinct viewer identifiers. It is only required to compute $3R$ inverse transforms per source frame, and R depends only logarithmically on N. Making $3R = 3\lceil \log_3 N \rceil = \Theta(\log N)$. □

The following theorem is devoted to per-frame watermarking complexity.

Theorem 3.2.10. *Let a video frame contain P pixels, and let N be the number of supported viewers. Let the scheme use $R = \lceil \log_3 N \rceil$ regions and $c = 3$ component variants per region. The per-frame pipeline performs:*

(i) one forward 2D FFT of the whole frame;

(ii) watermark embedding by modifying frequency-domain coefficients, creating components;

(iii) passing $3R$ component variants through full-frame inverse 2D FFT.

Then the per-frame computational complexity is

$$\Theta(P \log P \cdot \log N).$$

Proof. A forward 2D FFT over P pixels has complexity of $\Theta(P \log P)$. Watermark embedding costs are $\Theta(P)$ as stated in Lemma 3.2.8. By Lemma 3.2.9, the number of inverse transforms per source frame is $\Theta(\log N)$. Under the same assumption that forward 2D FTT the full-frame inverse FFT costs the same of $\Theta(P \log P)$, hence all inverse transforms cost

$$\Theta(\log N) \cdot \Theta(P \log P) = \Theta(P \log P \cdot \log N).$$

All main steps are independent of one another, and the sum of complexities is valid here. Summing the contributing steps yields the final complexity of

$$\Theta(P \log P) + \Theta(P) + \Theta(P \log P \cdot \log N) = \Theta(P \log P \cdot \log N).$$

□

Remark 3.2.11. The FFT and inverse FFT complexities are stated using $\Theta(\cdot)$ and not $O(\cdot)$ to reflect a tight asymptotic bound. Under the standard RAM arithmetic model, any algorithm that computes a full discrete Fourier transform must perform at least $\Omega(P \log P)$ operations; the same bound holds for the upper bound.

$\Theta(\cdot)$ assumes bounded-precision arithmetic and a standard RAM model, under which the FFT has a tight $\Theta(P \log P)$ complexity for full-frame transforms.

Copying operations are required to produce multiple copies of the original frame transform-domain frequencies for watermarking. Because the copying factor grows strictly more slowly, it does not affect the asymptotic complexity and is therefore omitted.

Empirical validation

To complement the complexity analysis, this section presents empirical benchmarks of the watermarking pipeline. The goal is to validate theoretical analysis against the relative costs of individual stages in practical implementation.

Each benchmark corresponds directly to a step analyzed in the complexity model: forward FFT, frequency-domain copying, watermark embedding, inverse FFTs, and total per-frame processing time. Later in the table, they are named as FFT, D2D, WM, and IFFT.

All timings were measured per processed frame and averaged over multiple attempts/frames to reduce noise. Measurements were obtained under identical conditions for setup, hardware, and software. Further details on the hardware and implementation are provided in Chapter 3.3.

To be clear, the following benchmarks cover only the watermarking computational costs of the analyzed steps. For example, encoding, normalization, loading and unloading data from memory, and other overhead are excluded and don't affect the steps discussed. This evaluation assesses only algorithmic processing complexity, not end-to-end system latency or system I/O transfer costs, which are independent of the algorithm.

Due to the nature of the implementation, all calculations are performed on the GPU using CUDA, which parallelizes most of the work performed by the watermarking steps. Parallelization affects only wall-clock execution time and does not impact the total amount of computational work performed. Therefore, the complexity analysis was expressed in terms of work complexity (Big-O / Θ) - counting the total number of arithmetic and memory operations, rather than execution time or memory usage.

As established in the complexity analysis, the computational work of the watermarking algorithm depends on the number of pixels, P, and the number of supported viewers, N. The relative runtime trends observed as P and N vary reflect the corresponding increase in computational work.

Therefore, the following experiments evaluate runtime scaling with respect to: **Number of viewers** N, or **Frame size** P.

Number of viewers N

Table 2 summarizes the mean per-frame runtime of each step of the watermarking pipeline for a fixed frame size P, measured across different numbers of regions R, which corresponds to viewer counts $N = 3^R$. The total pixel count P was fixed with value of 2073600, and averaging was performed over 1000 measurements/frames. These measurements provide an empirical basis for evaluating the theoretical $\Theta(\log N)$ scaling derived in the complex analysis Lemma 3.2.9 and later used in Theorem 3.2.10. Unprocessed CSV data are present in the "Big O analysis of N" folder, alongside the Jupiter notebook used to generate the tables and figures.

Table 2. *Mean per-frame runtime for fixed frame size P as a function of the number of regions R and viewers $N = 3^R$. All the runtime are given in (ms).*

R	N	FFT	D2D	WM	IFFT	Total	Overhead
1	3	0.411	0.156	0.088	0.862	1.876	0.359
2	9	0.293	0.267	0.092	1.229	2.437	0.555
4	81	0.268	0.517	0.138	2.188	4.076	0.964
6	729	0.242	0.731	0.176	2.824	5.230	1.256
8	6561	0.222	0.999	0.208	3.450	6.493	1.615
10	59049	0.208	1.224	0.239	3.917	7.596	2.007
12	531441	0.195	1.378	0.266	4.378	8.498	2.282
14	4782969	0.190	1.538	0.285	4.690	9.207	2.505
16	43046721	0.179	1.701	0.318	5.272	10.483	3.012
18	387420489	0.186	1.911	0.377	5.888	11.763	3.399
20	3486784401	0.174	2.110	0.384	6.429	12.733	3.636

The total time does not add up for individual pipeline stages due to execution overhead between GPU operations. This overhead includes kernel launch latency, synchronization (note that large-scale parallelization is occurring), and runtime bookkeeping (measuring the time of each stage adds time due to measurement being an operation). As the number of regions R increases, the number of GPU operations also increases, thereby increasing the overhead. Note that the expected operation count should grow in log N scale, and the overhead growth follows this; it does not grow linearly with respect to N.

Another nuance arises in FFT times, they should stay the same. The most plausible explanation is DVFS (Dynamic Voltage and Frequency Scaling). The GPU may start at a higher clock speed because it does not yet know the workload; sustained workload keeps or increases the clock (boost). However, at low workloads, it may be reduced.

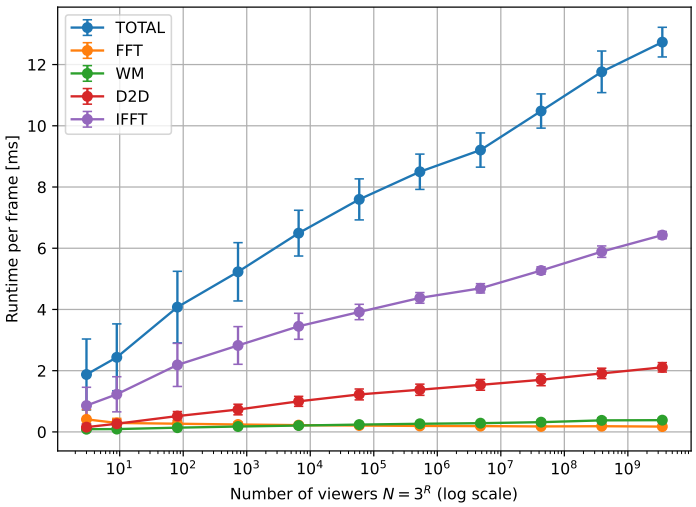
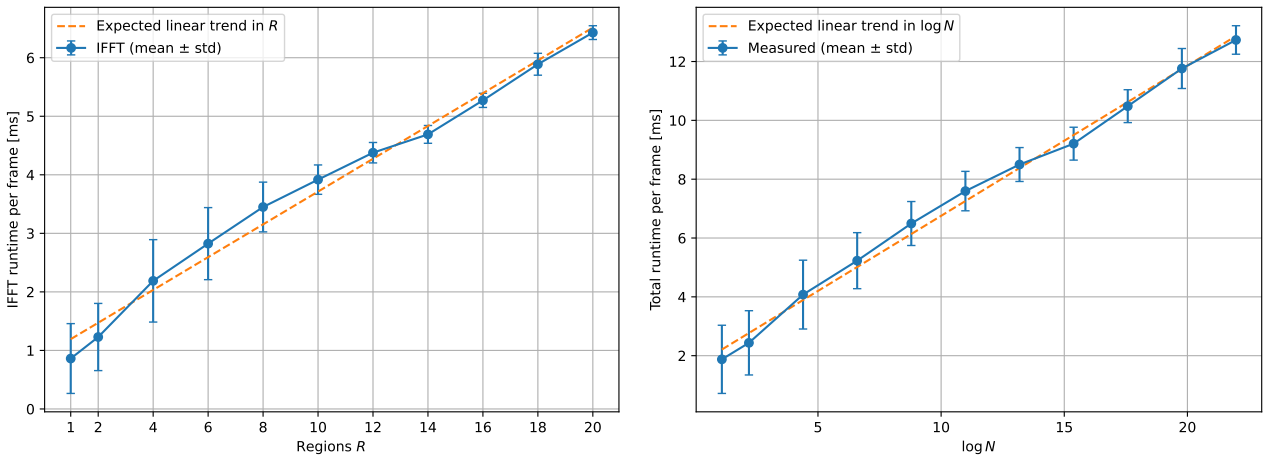


Figure 6. *Runtime scaling with the number of viewers* $N = 3^R$.

Figure 6 shows how the per-frame runtime scales with the number of supported viewers N for a fixed frame size P, note this is in log scale. As N increases, the total runtime grows approximately linearly with $\log N$, which is consistent with theoretical analysis.

Lemma 3.2.9 states that the number of IFFT operations is $3R = \Theta(\log N)$ as the N increases. Although the IFFT itself has a complexity of $\Theta(P \log P)$, with a scalable model introducing scaling through N it becomes $\Theta(\log N \cdot P \log P)$, since P is fixed, constants are dropped, leaving this with only complexity of $3R = \Theta(\log N)$. This is precisely visible in Figure 7a: the expected trend is plotted against the computational time required. Since the expected curve aligns with the actual curve, this empirically validates the theoretical structure. Time in the ms does not matter; only the ratios and differences between times across regions R, derived from N, are relevant. Additionally, the second Figure 7b validates Theorem 3.2.10, since the dominated cost is IFFT, and the theorem's main complexity comes from it.



(a) *IFFT runtime vs regions* R. ***(b)*** *Total runtime vs* $\log N$.

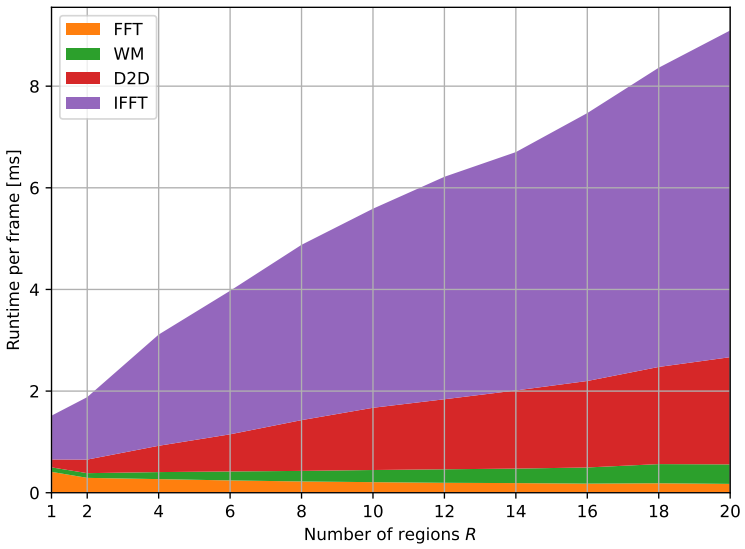
Figure 7. *Empirical validation of* $\Theta(\log N)$ *scaling for fixed frame size* P.

Since computational complexity is dominated by the IFFT, as shown in Theorem 3.2.10, it does

not make other pipeline steps free. To validate other pipeline stages, let us start with the simplest one, FFT. FFT is performed once per source frame, its complexity is stable and independent of N, and has a complexity of $\Theta(P \log P)$. Figure 8b supports this claim, as shown, the relative runtime of FFT is very stable and clearly independent of $R \approx \log_3 N$. The next step is WM, the watermark insertion step; it's the same as with FFT and is independent of N, with complexity $\Theta(P)$. Looking at the same Figure 8b, it is visible that there is a slight increase in runtime as R grows, this is because more smaller calls happen, so there is more system/hardware overhead happening than a single operation of same amount of work, overall the trend is stable and clearly does not meaningfully grow as N increases. D2D was not explored extensively, but for a good reason: it is highly dependent on implementation and hardware; this will be explored further in the next chapter. In terms of complexity, it is $O(P \log N)$: it is needed to copy P frequency bins $\log N$ times, noting that P is fixed in this experiment. The same Figure 8b supports this; the time clearly grows, and not in terms of calls overhead but $\log N$. IFFT has already been explored, so little to add here. Additionally, the percentage distribution of total time of each step is shown in Table Figure 8a, noticing that IFFT takes the most time and D2D is the runner-up.

R	FFT	WM	D2D	IFFT
1	27.1%	5.8%	10.3%	56.8%
2	15.6%	4.9%	14.2%	65.3%
4	8.6%	4.4%	16.6%	70.3%
6	6.1%	4.4%	18.4%	71.1%
8	4.5%	4.3%	20.5%	70.7%
10	3.7%	4.3%	21.9%	70.1%
12	3.1%	4.3%	22.2%	70.4%
14	2.8%	4.3%	22.9%	70.0%
16	2.4%	4.3%	22.8%	70.6%
18	2.2%	4.5%	22.9%	70.4%
20	1.9%	4.2%	23.2%	70.7%

(a) *Percentage contribution of runtime components per frame*



(b) *Per-frame runtime breakdown vs. number of regions R.*

Figure 8. *Watermarking pipeline runtime breakdown by stages.*

To conclude, by these runtime evaluations - Theorem 3.2.10 in terms of complexities regarding N theoretical basis is strongly supported by achieved results. On their own, these experiments do not prove the complexities; they only show trends, which is why the theorem was proven and is based on theory.

Frame size P

Table 3 summarizes the mean per-frame runtime of each step of the watermarking pipeline for a fixed number of supported users N, measured across different numbers of resolutions, which have different pixel counts P. The total number of viewers, N, was fixed at $R = 10$, making $N = 59049$, and averaging was performed over 1000 measurements/frames. These measurements provide an

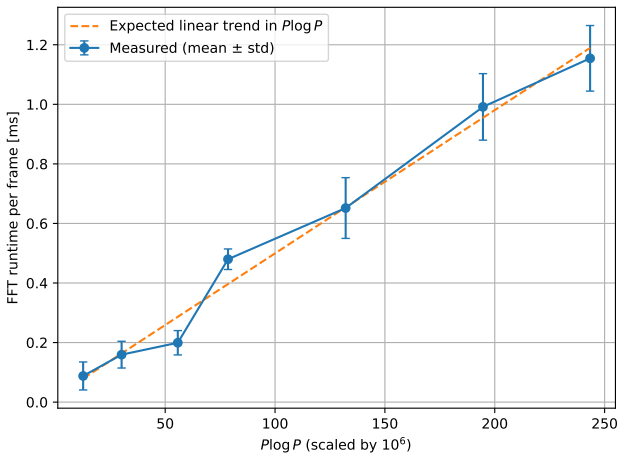
empirical basis for evaluating the theoretical $\Theta(P \log P)$ complexity, which dominates the final complexity in Theorem 3.2.10. Unprocessed CSV data are present in the "Big O analysis of P" folder, alongside the Jupiter notebook used to generate the tables and figures.

Table 3. *Test video resolutions and aggregated runtime (ms) results*

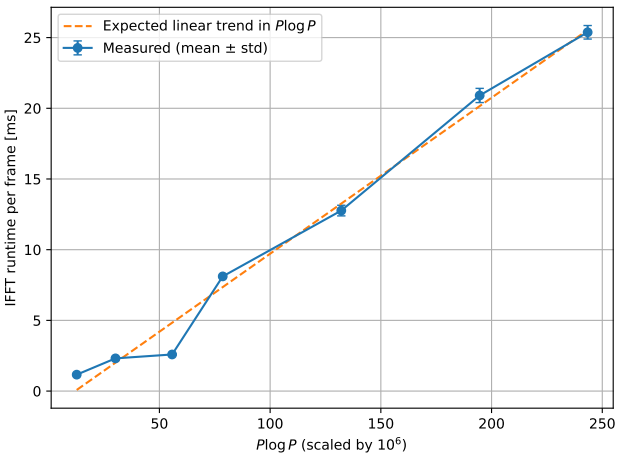
Name	Resolution	MP	FFT	WM	D2D	IFFT	Total
720p	1280×720	0.9 MP	0.088	0.261	0.808	1.161	3.636
1080p	1920×1080	2.1 MP	0.159	0.308	0.904	2.313	4.991
1440p	2560×1440	3.7 MP	0.199	0.417	1.917	2.589	6.526
5 MP	3008×1692	5.1 MP	0.48	0.482	2.412	8.111	13.212
4K	3840×2160	8.3 MP	0.652	0.697	3.479	12.766	19.529
12 MP	4608×2592	11.9 MP	0.991	1.018	5.167	20.906	30.23
5K	5120×2880	14.7 MP	1.154	1.251	6.343	25.375	35.877

Literature on FFT has already established the complexity of FFT and IFFT being $O(P \log P)$. Figures 9a, 9b partially verify this, partially because the $\log P$ factor is not really visible here. This is because the pixel range is too small to show a meaningful impact of $\log P$, and the linear part of P dominates it. The pixel range was chosen to include standard 16:9 resolutions used in practice, as these are expected for the proposed models' applications. Regarding the WM stage, as it should have complexity of $\Theta(P)$ as Lemma 3.2.8 showed, the WM runtime is plotted against P in Figure 9c and follows the expected trend. The final Figure 9d verifies that complexity as Theorem 3.2.10 states of $\Theta(P \log P \cdot \log N)$, since while N is fixed, the only remaining part is $P \log P$, as this is the curve plotted. Once again, the $\log P$ term cannot be fully verified, but we can verify that it's not the next complexity step, which would be $\Theta(P^2)$.

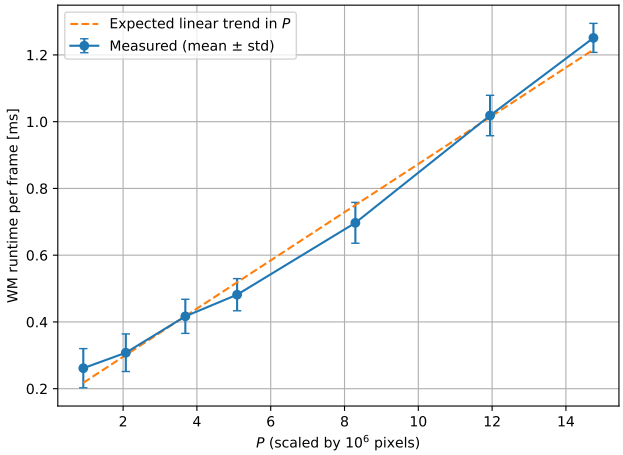
There is a little dip at the third data point regarding FFT/IFFT/Total. This is likely due to software implementation, as 1440p has a high multiple of 2 (being a multiple of 2 is crucial for good performance, relating to FFT theory) and/or CUDA offers better optimization for this widely used resolution.



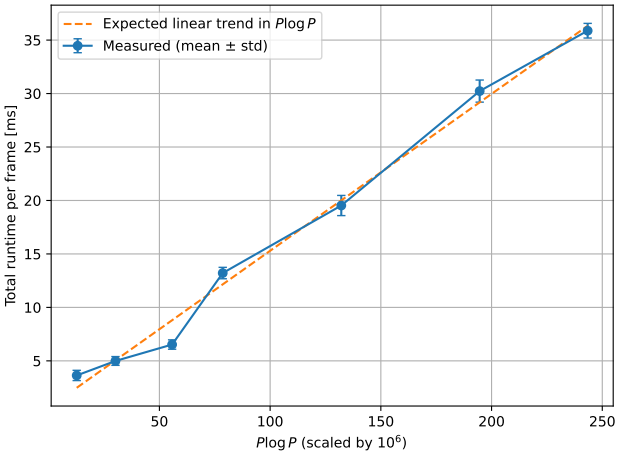
(a) *FTT runtime vs* $P \log P$.



(b) *IFFT runtime vs* $P \log P$.



(c) *WM runtime vs* P.



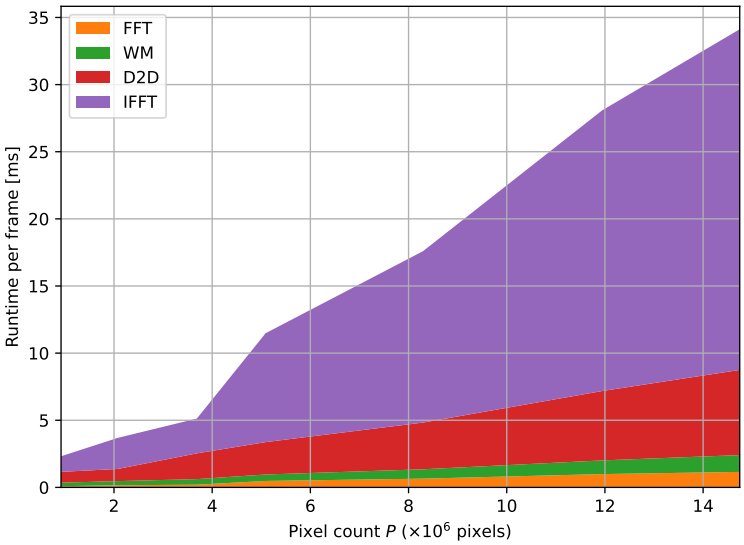
(d) *Total runtime vs* $P \log P$.

Figure 9. *Empirical validation of* $\Theta(P \log P)$ *in terms of* P*, for fixed* N.

Figure 10b beautifully illustrates that all of the stages in the pipeline are related to P. Reminding that FFT is performed only once, and WM is quite inexpensive and has no constant as a scaling factor related to N. Going to the Figure 10a, IFFT and D2D remain the costliest, especially IFFT. The same trend was also visible in the previous experiment, Figure 8a. This is to be expected, since that step generates the unique variants that enable scaling, allowing us to support N viewers.

MP	FFT	WM	D2D	IFFT
0.9	3.8%	11.3%	34.9%	50.1%
2.1	4.3%	8.4%	24.5%	62.8%
3.7	3.9%	8.1%	37.4%	50.5%
5.1	4.2%	4.2%	21.0%	70.6%
8.3	3.7%	4.0%	19.8%	72.6%
11.9	3.5%	3.6%	18.4%	74.4%
14.7	3.4%	3.7%	18.6%	74.4%

(a) *Percentage contribution of runtime per MP.*



(b) *Per-frame runtime breakdown vs.* P.

Figure 10. *Watermarking pipeline runtime breakdown by stages.*

3.2.3 FFT pruning and other theoretical optimizations

In many practical signal-processing pipelines, only a subset of spatial-domain samples is required after frequency-domain processing. To reduce IFFT computation time, a pruned IFFT can be used, as performing a full one is unnecessary. In the literature, this is also referred to as the partial, truncated, selective, or pruned inverse FFT. The idea is to compute only the subset of spatial-domain samples required, rather than the whole frame. This is precisely what is already partially done: $1/R$ of the pixels are only encoded, resulting in clear waste.

Because we only need $k = 1/R$ output pixels from P, the current full IFFT is inefficient. The simplest way to understand how this could be achieved is to note that the FFT/IFFT can be implemented recursively; a pruned version would drop branches that don't contribute to those k outputs. In essence, performing only the necessary work to reconstruct the samples required.

Although the inverse FFT can, in theory, be pruned in the same way as the FFT, practical applications of pruned IFFT are very limited or non-existent. the vast majority of research has focused on only pruned FFTs, since most real systems require only a subset of frequency-domain outputs rather than a subset of time-domain samples. There is little to be gained even by pruning FFT, as the complexity is at best $O(P \log k)$. While the complexity is reduced, it is almost irrelevant for most of the practical input sizes, as the $\log k / \log P$ part grows extremely slowly compared to the linear-P part. Additionally, most applications operate on 1D signals, where the FFT is most commonly used; however, for our purposes, a 2D scenario is more advantageous for pruning.

To analyze the potential reduction in complexity, we examine how the 2D FFT is typically computed. There are actually two ways to do that Column/Row or in opposite order Row/Column:

- Do H FFTs for each row W, complexity - $O(H \cdot W \log W)$
- Do W FFTs for each column H, complexity - $O(W \cdot H \log H)$

So in total:

$$O(H \cdot W \log W + W \cdot H \log H) = O(H \cdot W(\log W + \log H)) = O(H \cdot W \log(H \cdot W)) = O(P \log P)$$

As IFFT performs a symmetrical but ”opposite” FFT operation, the same complexity and steps are present.

This is very beautiful and simplifies our work; we can perform a full row-IFFT and then prune the column-IFFT outputs. Row IFFTS for all H rows is $O(H \cdot W \log W)$, and for each of W columns - $O(W \cdot \frac{H}{R} \log H)$.

$$O(H \cdot W \log W + W \cdot \frac{H}{R} \log H) = O(P \log W + \frac{P}{R} \log H)$$

Lets explore couple of resolutions 1080p - 1080×1920, $W = 1080$, $H = 1920$ and 5K 2880×5120, $W = 2880$, $H = 5120$, and for regions lets say we have $R = 10$.

- Standard complexity computations 1080p - $P \log_2 P = 1080 \cdot 1920 \log_2 1080 \cdot 1920 = 4.3 \cdot 10^7$. 5K - $P \log_2 P = 3.5 \cdot 10^8$.
- New theoretical complexity computations 1080p - $P \log_2 W + \frac{P}{R} \log_2 H = 2.3 \cdot 10^7$, 5K - $... = 1.9 \cdot 10^8$

Using $\log_2$ over $\log_e$ is valid in FFT since it usually has radix-2 stages, so the number of stages is $\log_2 P$.

Returning to the comparison of operation counts, we observe that for the 1080p variant, $\frac{2.3 \cdot 10^7}{4.3 \cdot 10^7} = 0.53$, indicating that, under current assumptions, we reduce the operation count by nearly half. A similar observation holds for the 5K variant: $\frac{1.9 \cdot 10^8}{3.5 \cdot 10^8} = 0.54$.

Depending on the video orientation, this pruning can be done either on height or width. The most significant dimension should always be pruned, as it yields the most tremendous improvement.

To summarize, pruning the IFFT in this way has strong potential to reduce computational workload and to achieve faster watermarking pipeline runtimes. As we saw in Theorem 3.2.10 and will see in a later chapter, the IFFT is the most computationally expensive step; reducing its complexity is crucial.

Funnily enough, even with this improvement, there is still some unnecessary work within the IFFT, and it can be improved further. Let us remember that IFFT complexity also scales with N in a $\log N$ term. We perform similar work multiple times across all components of the IFFT; only a small portion of the bins are unique. Only the number of bins touched by watermarking steps differs across IFFT attempts, leaving $\frac{R-1}{R}$ of the bins unchanged per region.

Let’s look at a concrete example. To better understand the concept, let us use a 1x4 image with 4 regions, 1 pixel per region. Typically, IFFTs are performed over the entire frame; however, since IFFTs are similar, we can cache partial results. For the first component, the standard full-frame IFFT is performed; for component one, only the bin is changed, and cached calculations can be reused for the third component in the same region. Moving to the next region, 2 of 4 bins are already included in the partial results; this applies to all components. It is better to compute the full unwatermarked frame IFFT and store the partial calculations, then use them as needed.

Caching and IFFT pruning can severely reduce the computational complexity of the watermarking pipeline discussed. They are both very promising and offer substantial improvements, probably enough to vanish $\log N$ term from the final complexity presented in Theorem 3.2.10 of $\Theta(P \log P \cdot \log N)$ while also improving on the $P \log P$.

3.3 Implementation

The next step to evaluate the suggested model, which was presented in 3.2 is to implement it practically. Videos are converted to grayscale to simplify the process, having only one band for color. To efficiently and quickly convert between the frequency domain via FFT and the spatial domain via IFFT, these operations are performed on the GPU. Since GPUs have many cores, a high-bandwidth architecture, and a better latency-hiding scheduler, they are far superior to CPUs for large-scale IFFT applications. Compute Unified Device Architecture (CUDA) is NVIDIA's parallel programming model that enables us to perform the required operations. All of code base is present

3.3.1 Hardware and software environment

Table 4. *Hardware environment used in development and experiments.*

Component	Specification
CPU	AMD Ryzen 9 7900 (12-core, 24-thread, base clock: 4.7 Ghz)
GPU	NVIDIA GeForce RTX 4070, 12GB VRAM
RAM	32 GB, Dual-channel (DDR5-6000 CL30)

Table 5. *Software environment used in development and experiments.*

Component	Version
OS	Windows 11
CUDA	Toolkit 12.9; NVIDIA Driver 580.97 (Studio)
Java	24.0.1 (Oracle OpenJDK)
JavaCV	1.5.12
FFmpeg	git-cdbb5f1b93 (2025-08-14), GCC 15.2.0 (MSYS2).
Encoding Threads	8 GPU(limitation by NVIDIA drivers), ∞CPU

3.3.2 System overview

Before processing the video stream, some prep work is done, which includes loading the watermark and frame parameters from Java to the C++ side. This flow is presented in Figure 11. First, it starts from "1." C++ side is prepared, which creates and allocates workplaces and plans for R2C (Real to Complex) and C2R (Complex to Real). Additionally, since here the watermark is static, its data is also loaded. Next, pinned memory direct buffers are allocated and shared between C++ and Java

sides for more efficient memory management. This improves memory management and reduces redundant data. This is the same software that was used to generate

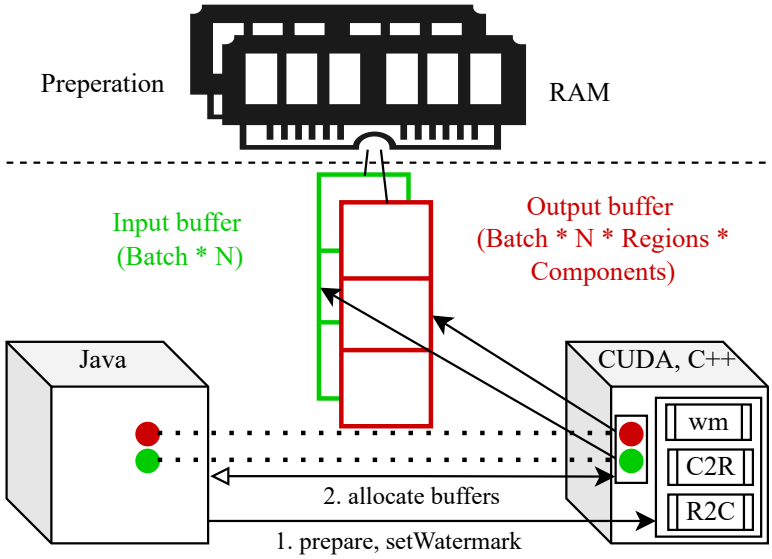


Figure 11. *Preparation code flow before watermarking.*

Having a workplace prepared process begins with JavaCV's `FFmpegFrameGrabber`, which decodes frames into RAW pixel data, with multiple color channels. Frames are then staged in the input direct buffer. Once enough frames (depending on the Batch setting) are loaded, watermarking can be performed on the GPU. The watermarking flow is illustrated in Figure 12. Firstly, see (1) in the image, the loaded input direct buffer is loaded into the GPU's VRAM via H2D (Host to Device). After that, see (2), the R2C plan (prepared prior, in preparation step), is used to perform FFT on the loaded frames data that's loaded on VRAM. The output is frequency domain data, which is then copied in step (3); each copy receives a unique watermarked region applied. In step (4), each watermarked version gets converted back to the spatial domain by performing IFFT with the C2R plan. Lastly, in step (5), D2H (Device to Host) is called to copy watermarked frames from VRAM to RAM-pinned memory direct buffer. This buffer is accessible from the Java side, which now encodes all the frames.

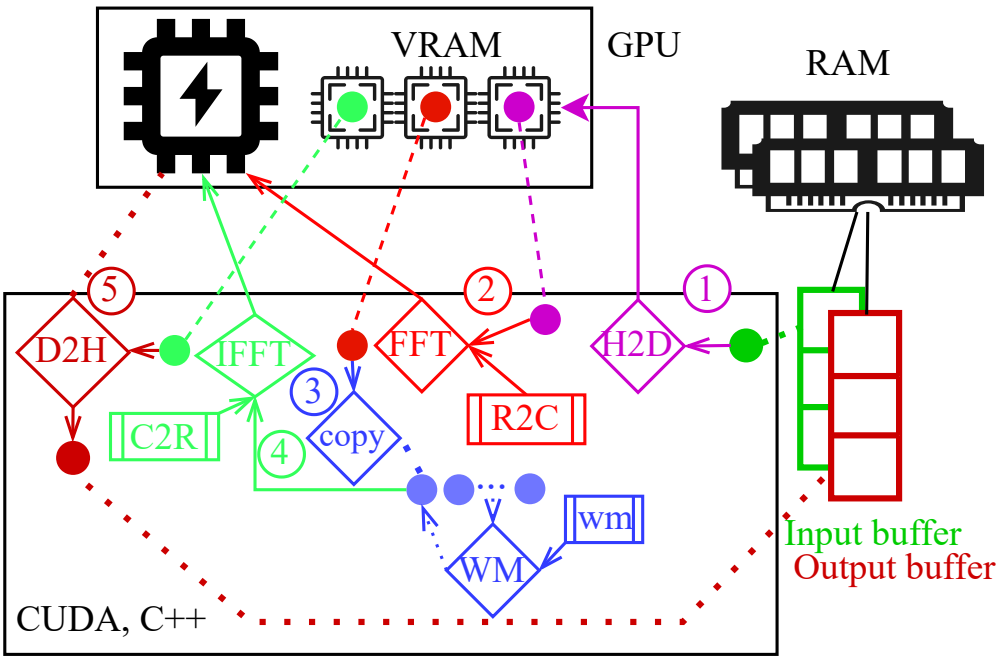


Figure 12. *CUDA code flow, watermarking with GPU.*

For code representation of what was presented here, check out the folder ”instructions” and the video ”Explaining main code.mp4”, which goes over when this code is executed and shows precisely where each of the steps is implemented.

3.3.3 Optimizations, parallelization, and other

On the CPU, watermarked versions encoding is done in parallel. Frames are sent in batches to be watermarked to reduce the JNI (Java Native Interface) calls overhead. On the GPU, occupancy is tuned by (blockSize, gridSize); shared memory tiles avoid redundant global reads. FFT workplaces and plans are reused across frames, since resolution and other parameters don’t change. There is still room for improvement; the most significant speedup was achieved through efficient memory management, limiting buffers to bytes and floats, and avoiding double precision since it’s unnecessary.

3.3.4 Watermarking flow in detail

To further understand how watermarking looks in practice, refer to illustration Figure 13. As discussed before, firstly, our frame goes through FFT, resulting in a frequency domain representation of the same frame. Half of the values remain the same via reflection and the FFT property, so we only have half of the frequency domain to work with, as the other half will undergo the same operations. Inside the frequency domain, it’s crucial to avoid watermarking low frequencies since they risk visible artifacts back in the spatial domain. After removing low frequencies from the watermarking area, we split it into as many regions as we have. In this example, we only have four regions. Now the watermarking process can begin, starting with copying the whole frequency domain, applying one of the selected watermarks to a specific region only, and repeating this process to get each component.

For this example, a simple watermark variant design is chosen, as seen in Figure 14. After applying watermarks to each of the frequency domain copies, the next step is performing the inverse FFT to convert the frame back to the spatial domain. After storage and encoding optimization, only pixel regions will be used in the final broadcast, leaving only slices of the original. In this example, we have 4 regions with 3 watermarks/components in each region, for a total of 12 components, and it supports at most $3^4 = 81$ viewers.

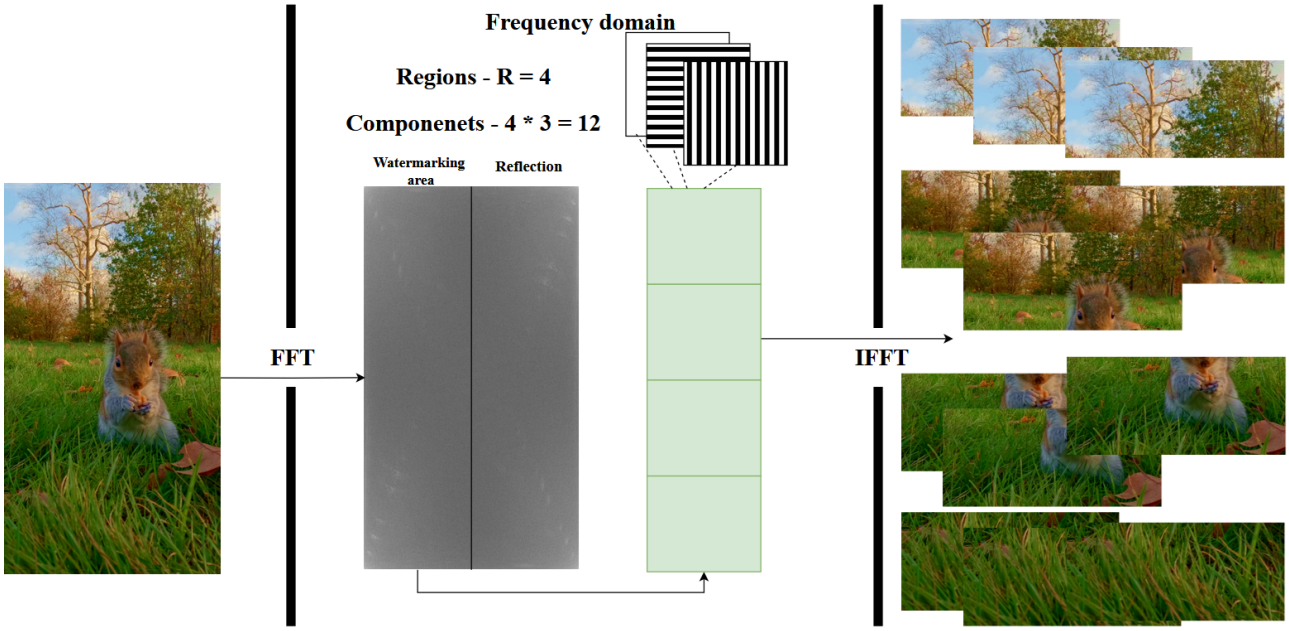
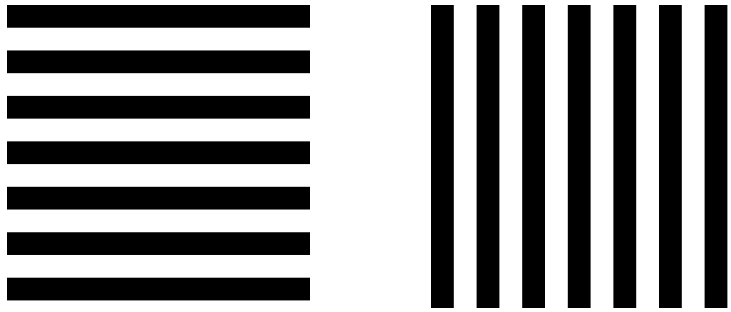


Figure 13. *FFT watermarking flow*

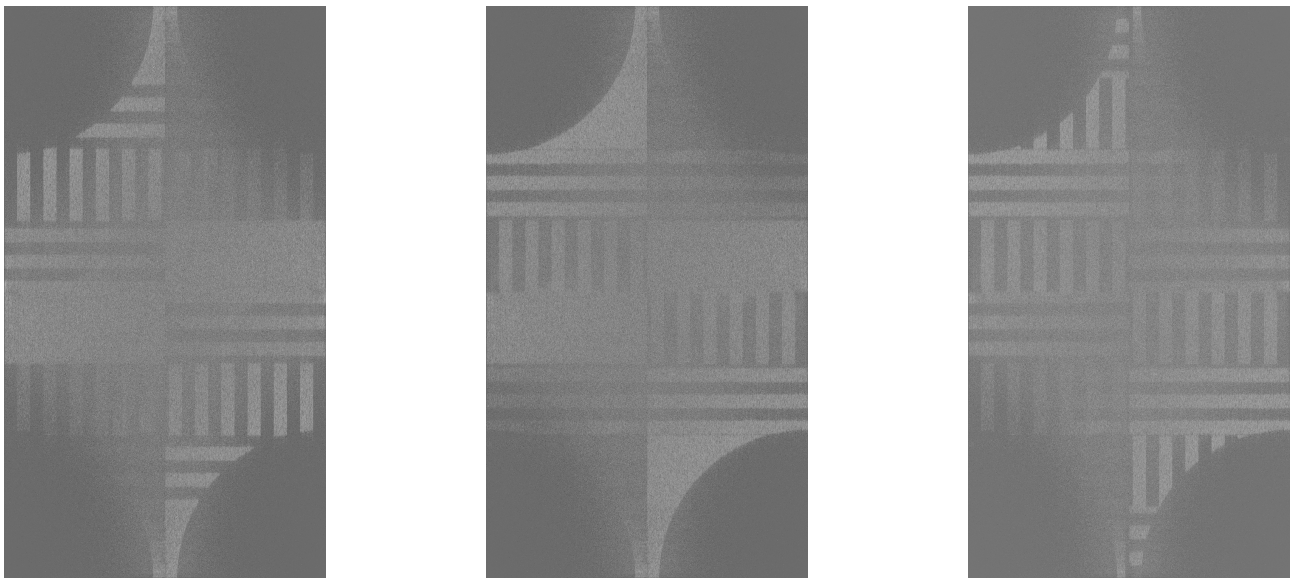


a) Fully white b) Horizontal lines c) Vertical lines

Figure 14. *Three base watermarks: vertical, full, horizontal*

3.3.5 Watermark extraction

After combining multiple components back in the spatial domain, we get unique variants. To extract the watermark, we go through a forward FFT with our unique variant and original videos. Subtracting and getting the difference between them reveals the watermark, for example, which can be seen in Figure 15. Here, looking at the Figure 15(a), a variant of A-B-C-B-A-C-B-A; in this case, the viewer would be identified. This was extracted from a single frame, averaging over multiple frames would yield clearer extraction, higher confidence even though here it is enough as is.



(a) *A-B-C-B-A-C-B-A* ***(b)*** *A-A-B-C-A-B-B-A* ***(c)*** *B-C-B-C-B-C-B-C*

Figure 15. *Extracted watermarks, image should be interpreted left side up to down, or start from the right side and go from bottom to the top.*

All video components (.mp4 files) are present in the folder ”Watermark-examples”, along with the settings used to create them.

3.3.6 Benchmarking

To evaluate the model's real-time viability, benchmark metrics were used. Input video data are listed in Table 6. This same video will be watermarked with a different number of regions, R. Just reminding that $R = \lceil \log_3 N \rceil$ and the other way around $N = 3^R$, where N represents the count of possible unique variants. Three regions, each with counts of 5, 8, and 10, were benchmarked, where each can support at most - $3^5 = 243$, $3^8 = 6\,561$, $3^{10} = 59\,049$ viewers.

Table 6. *Video parameters*

Parameter	Value
Name	street.mp4
Resolution	1080×1920
Video length	59.60 s (60 FPS)
Total frames	3576

Let us begin by defining the evaluated work, which comprises sequential stages in the pipeline, from capturing the frame to outputting the components (partial videos). There are three main stages:

- Convert - prepare frame: primarily involves decoding and writing pixel data to the input byte buffer. What could be done to achieve faster times: reduce work by leveraging native handling and/or parallelization, having two input buffers so loading ”work” never stops.

- GPU - Figure 12 covered main work performed by that step, high level this step does FFT -> watermark insertion -> IFFT. What could be done for faster times: reducing complexity by following the suggestions presented in Chapter 3.2.3.

- Encode - work being done is the same as the name, encoding the final components so they can be stored, or prepared for broadcasting. Possible optimizations: increasing the number of GPU encoding threads, as only eight components are currently encoded on the GPU, with the remainder processed on the CPU, which can create bottlenecks and slower processing times. Currently, because consumer-grade NVIDIA drivers limit encoding threads to 8, this limit is artificial, as the card can support many more. This limit can be disabled by "hacking" the drivers, but as this is illegal, it will not be covered or used here. Data center GPUs are not limited and are well-suited for real commercial applications of this model.

Having established the stages, I would also like to explain why optimizations were included to demonstrate that there remain realistic and meaningful improvements that would reduce the final time. This indicates that the following benchmarks are not performance limits and that faster results are achievable.

Let us begin by noting that the Convert step is the least expensive and relatively stable across the different region attempts. This is easily explained: across attempts, the total work done in that step remains constant; variation over time is not marginal; and the difference arises from the environment being highly influenced by how the code runs, how much of the CPU is available at each time, and similar factors. This phenomenon is well known as System-level performance nondeterminism.

It is also clear that the most expensive step is Encode. This step is not fully parallelized because insufficient GPU or CPU threads are available, so some virtual CPU threads are created, resulting in suboptimal parallelization. Overall, the times starting from $\approx$ 31 seconds for 5 regions and $\approx$ 40 seconds for 10 regions show that the work is not doubled. Actually, what Lemma 3.2.6 suggested was that the encoding time should be constant as N increases/decreases, which was not fully achieved due to not having proper hardware to see that. Even with appropriate hardware, there would still be some overhead from increased calls, but it would not be significant in the big picture. Overall, the time differences across regions are reasonable and consistent with expected scaling.

GPU time is the runner-up in cost. An in-depth breakdown of all substantive steps in that stage is presented in Table 8. For now, the GPU time grows the quickest as the N increases; this is completely to be expected, as most of the required steps are influenced by the complexity $\Theta(\log N)$. Complexity was explored and shown in Theorem 3.2.10.

Total time is what determines this model's practical viability for real-time applications. Since we are evaluating for real-time applications, note that the original video was 60 fps and 59.6 seconds long, as reported in Table 6. To make this work per source frame, the pipeline should have the speed of $\frac{1000 \text{ ms}}{60} = 16.666$ ms per frame. For the 5 regions, $15.96 \leqslant 16.67$ is real-time viable; for regions 8 and 10, $18.72 \nleqslant 16.67$ and $21.01 \nleqslant 16.67$, the required time is higher and therefore not feasible under current hardware/software. Although the time is not within the required range, it remains reasonable and does not increase by an order of magnitude.

Table 7. *Runtime breakdown for different numbers of regions (3 components). Times are in ms.*

	5 Regions		8 Regions		10 Regions	
Stage	**Total**	**Per frame**	**Total**	**Per frame**	**Total**	**Per frame**
Convert	6 588.75	1.84	6 253.53	1.75	6 906.34	1.93
GPU	13 108.63	3.67	18 284.05	5.11	21 963.79	6.14
Encode	31 285.59	8.75	36 507.33	10.21	39 947.47	11.17
Unaccounted	6 104.27	1.71	5 887.80	1.65	6 315.49	1.77
Total	**57 087.24**	**15.96**	**66 932.70**	**18.72**	**75 133.09**	**21.01**

All benchmark results and video components (.mp4 files) created during the test are present in the folders ”Real time benchmark X Regions”, where $X = 5, 8, 10$, so in total three folders, along with console outputs and the settings used to create them.

GPU time is dominated, as expected, by IFFT and the main watermarking steps. However, there are some direct hardware costs, such as H2D, D2H, and D2D. These times are highly dependent on hardware. This included PCLe/NVLink bandwidth & latency, system ram speed, pinned or pagable memory(here we use pinned so better), Non-Uniform Memory Access topology, and even on the GPU copy engine count. This is well beyond the discussed subject, want to state that results/times are highly dependent on hardware, and the used hardware is not state-of-the-art for 2026.

Table 8. *Summary of GPU pipeline run-times (ms), for R=10 (c=3), averaged over 1000 source frames.*

Stage	**Total**	**Per frame**
Memory transfer		
H2D	331.95	0.33
D2H	1004.14	1.00
Core FFT pipeline		
FFT	223.77	0.22
D2D	1223.40	1.22
Watermark (WM)	388.22	0.39
IFFT	3636.71	3.64
Normalization	511.98	0.51
TOTAL	**9075.57**	**9.08**

GPU benchmark results gotten during the test are stored in folders named ”Real time benchmark GPU”, along with the console outputs and the settings used to generate them.

4 Visual Impact Analysis

To assess the practical viability of the proposed model, it is vital to evaluate its robustness and visual impact on the host medium. While robustness is necessary, excessive distortions introduced by the watermark would render subsequent analysis irrelevant. To objectively evaluate the performance of the proposed model, researchers already have some assessment metrics such as PSNR (Peak signal-to-noise ratio), SSIM (Structural similarity index measure), and VMAF (Video Multimethod Assessment Fusion), and others.

4.1 Peak Signal-to-Noise Ratio (PSNR)

Peak signal-to-noise ratio is one of the earliest objective quality metrics used in image and video processing. Although introduced in the 1970s, it remains one of the most widely used tests, although some improvements have been made since then. It measures the ratio of the maximum possible pixel value to the mean squared error (MSE) between the original and watermarked frames. If the original frame is O, and the watermarked is $\hat{O}$, where both of them are of sizes $W \times H$ (width and height, total pixels), then MSE is defined like this:

$$\text{MSE} = \frac{1}{HW} \sum_{h=1}^{H} \sum_{w=1}^{W} \big(O(h,w) - \hat{O}(h,w)\big)^2$$

Then PSNR (measured in dB) is finally found like this:

$$\text{PSNR} = 10 \log_{10} \left(\frac{MAX_P^2}{\text{MSE}} \right) \tag{1}$$

MAX_P represents the maximum possible pixel value. The most common value is 255, which corresponds to an image bit depth of 8 bits per channel. Not $256 = 2^8$ since 0 is also a possible value, total values are 256 in the range 0 - 255, but the maximum value remains 255.

4.1.1 PSNR-HVS and PSNR-HVS-M

Enhanced variants of the traditional PSNR metric are also used, PSNR-HVS and PSNR-HVS-M - both originally introduced by Ponomarenko et al. (see [4, 10]). PSNR-HVS better reflects the human visual system (HVS), enabling it to capture perceived distortions better. This is achieved by incorporating frequency sensitivity via contrast-masking models applied to the error signal. Further improved metric PSNR-HVS-M extends it further, by analyzing the business of the regions, whether they are smooth or heavily textured and complex.

This looks like this, every 8×8 blocks of original and watermarked frames are passed through DCT (discrete cosine transform), $A = DCT(O)$, $\hat{A} = DCT(\hat{O})$. Then the DCT-domain error is defined as $E_{hw} = A_{hw} - \hat{A}_{hw}$, and the HVS weights are applied to represent the fact that human eyes are less sensitive to high-frequency coefficients, which have lower weights, and main emphasis

is on mid frequencies, $E_{hw}^{`} = Weights_{hw} \cdot E_{hw}$. Same as before, now PMSE (perceptual mean squared error), not MSE, PMSE $= \frac{1}{HW} \sum (E_{hw}^{`})^2$ and finally PSNR $-$ HVS $= 10 \log_{10} \left(\frac{MAX_P^2}{\text{PMSE}} \right)$.

In PSNR-HVS-M, the core idea is to compute block activity, where higher activity (value) indicates complex/textured regions and lower activity indicates smooth areas.

In practice, PSNR-HVS weights low-frequency errors more heavily because they are less noticeable, and the further extension PSNR-HVS-M indicates that distortions in heavily textured and busy regions are less evident than in smooth areas.

4.1.2 Practical interpretation of PSNR, PSNR-HVS, and PSNR-HVS-M

Having established the theoretical background of the PSNR-based metrics, it is necessary to understand how to read these values in video-quality evaluation. Although each PSNR variant differs in how it assesses visual perception, they all aim to quantify the similarity between an original and a modified frame. The higher the value, the more similar the compared frames; lower values indicate apparent degradation.

Although PSNR measures visual fidelity rather than sound or energy, it is derived from the energy-based SNR (Signal-to-Noise Ratio). Since PSNR is expressed in decibels, it is a logarithmic transformation of the underlying error. This means that perceived ”distance” between values is not linear; a slight numerical increase in dB corresponds to a significant proportional reduction in error. Since PSNR is an extension of SNR, it remains measured in decibels. To familiarize with what this means in practice, here are some examples:

- 3 dB increase means the mean squared error (MSE) is halved.
- 6 dB increase means the MSE is reduced by a factor of four.
- 10 dB works out to an MSE reduction of a factor of ten.

For our purposes, going from 30 to 33 dB is twice as ”clean”, while going from 40 to 43 dB also means the same, even though the quality was already high. When the distortion becomes very small, improving the results requires enormous improvements in the watermarked version. For example, a video going from 26 to 30 dB will show a dramatic visual difference, but going the same 4 dB in 40 to 44 dB likely will look identical even though mathematically the error was reduced by the same factor of $\approx$ 2.5. The maximum value is considered infinity since MSE would be zero if frames are identical, and the limit of the expression in (1) tends to infinity.

This logarithmic compression is also why the PSNR value will rarely exceed $\approx$ 45 dB; the required reduction in error becomes extremely large. To evaluate the value ranges, the following guidelines are commonly used for interpreting quality in video watermarking and compression research.

General interpretation ranges:

- < 30 dB - distortions clearly visible; sharpness loss, flickering, ringing, clear noise is present.
- 30 - 35 dB - acceptable, but some visible artifacts are present, especially on static backgrounds.
- 35 - 40 dB - good quality, distortions are difficult to detect.

- > 40 dB - very high quality, likely no visible changes.

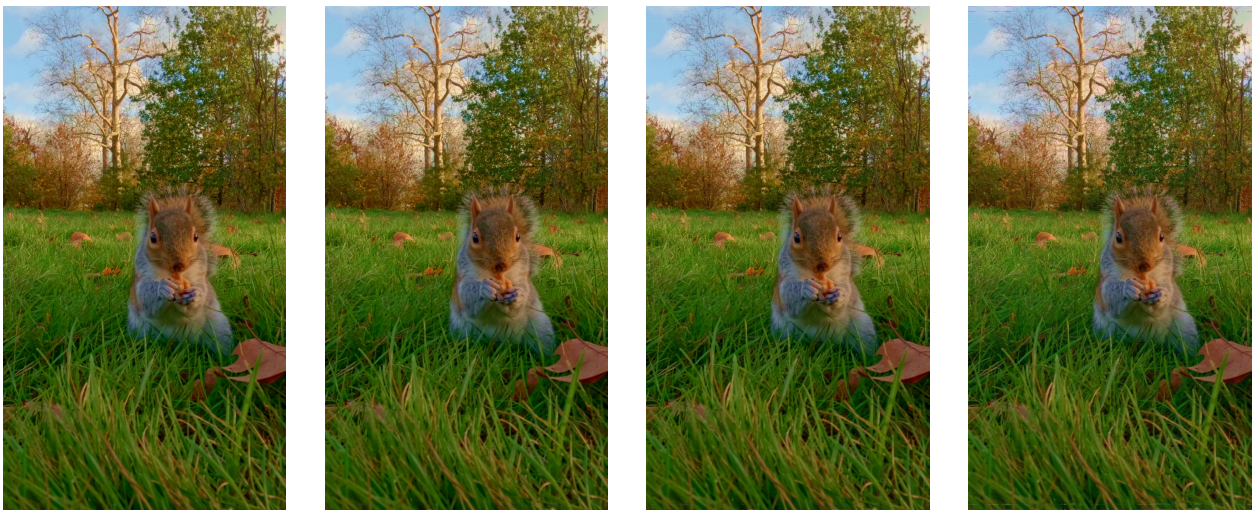
These ranges are only guidelines, and their interpretation also depends on the metric, as each captures a different aspect of human perception. PSNR reflects only the mathematical error, while PSNR-HVS and PSNR-HVS-M try to capture structure and textures. Differences among the metrics could also provide additional context on where and how the distortions are present.

For example, a high PSNR with a lower PSNR-HVS could indicate low global distortion, but the HVS model detects changes in features such as edges, flat backgrounds, and smooth areas (e.g., PSNR = 38, PSNR-HVS = 33). Another common scenario is having a significantly higher PSNR-HVS-M than the other two metrics, which could indicate that distortions are in busy or heavily textured regions, which would be invisible during playback (e.g., PSNR = 33, PSNR-HVS = 32, PSNR-HVS-M = 40).

To evaluate the efficiency of the watermarking model presented in Chapter 3.2, PSNR metric extraction methods were implemented in Java. All three PSNR variants (PSNR, PSNR-HVS, PSNR-HVS-M) are available in the codebase; path: src/main/java/org/example/tests/PSNR.java. The primary method is findPSNR, which expects two arguments (testPath and originalPath, both String) and outputs the three PSNR metric results to the console.

All computations are performed on the luma (Y) channel, which is standard practice for PSNR evaluation. Additionally, following the original formulation, both PSNR-HVS and PSNR-HVS-M are using non-overlapping 8×8 DCT blocks (without a sliding window) [10].

How PSNR looks in practice is shown in Figure 16 To obtain representative outputs, vSquirrelEating.mp4 (from media/inputs) was used to embed the watermark at different strengths. PSNR results for each of the strengths are shown in Table 9 A complete analysis, along with other metrics, is presented in Chapter 4.4. Figures and table shown here is only intended to present PNSR, as some crucial nuances and aspects are omitted making this data not representative of the watermarking model.



a) Original b) Strength 10 c) Strength 20 d) Strength 50

Figure 16. *PSNR examples with different watermark strengths*

Watermark Strength	PSNR (dB)	PSNR-HVS (dB)	PSNR-HVS-M (dB)
Original	$+\infty$	$+\infty$	$+\infty$
10	35.1	27.2	39.8
20	28.9	20.9	28.3
50	21.7	13.7	17.5

Table 9. *PSNR, PSNR-HVS, and PSNR-HVS-M values for different watermark strengths*

4.2 Structural similarity index (SSIM)

SSIM is a quality metric that measures the structural similarity between the original and distorted images. Unlike PSNR, which only considers pixel-wise error, SSIM accounts for changes in luminance, contrast, and structural patterns. To capture structural information in an image, SSIM decomposes image comparison into three components: luminance similarity, contrast similarity, and structural similarity. Similarly to PSNR-HVS, to capture structure, 8×8 or 11×11 window blocks can be used, yielding ESSIM (Edge-based SSIM). Let us represent them here as for the original patch - x, and for the watermarked y, SSIM is now defined as:

$$\text{SSIM}(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{2}$$

where

- μ_x,μ_y - mean luminance
- σ_o,σ_y - standard deviation (contrast)
- σ_{xy} - covariance (structural similarity)
- C_1, C_2 - stabilizing constants

$$C_1 = (K_1 L)^2, \qquad C_2 = (K_2 L)^2,$$

- where L denotes the dynamic range of pixel intensities and K_1 and K_2 are small empirically chosen constants.

SSIM values range from -1 to 1, where 1 indicates perfect similarity. The sliding window technique is used to evaluate each available block for ESSIM variant. For example, with a stride of 1 and a frame of 1920 ×1080, the total positions would be $(1920 - 8 + 1) \times (1080 - 8 + 1) = 2052649$. Local SSIM values are then averaged, creating a more accurate metric. Most libraries use a Gaussian-weighted window with a stride of 1 or 2.

4.2.1 Multi-Scale Structural Similarity (MS-SSIM)

Multi-Scale Structural Similarity (MS-SSIM) further builds on SSIM. It aims to capture more distortions across different image scales, such as blur, compression artifacts, and loss of fine detail. The

core idea is based on the human visual system's hierarchical perception of image quality. Evaluating structural similarity across different down-sampled versions of the original image should better capture what SSIM aims to achieve.

Proposed by Wang et al. [15], MS-SSIM uses a low-pass filter and down-sampling to construct a multi-scale image pyramid. At each step, the contrast and structural similarity components are computed (similar to SSIM). At the same time, the luminance comparison is applied at a larger scale to reflect the reduced sensitivity to absolute luminance differences at finer resolutions. The similarity measures obtained at different scales are then adjusted by empirically determined weights to reflect the relative perceptual importance of each scale.

Given two images x and y, MS-SSIM is computed over M scales as [15]:

$$\text{MS-SSIM}(x,y) = [l_M(x,y)]^{\alpha_M} \prod_{j=1}^{M} [c_j(x,y)]^{\beta_j} [s_j(x,y)]^{\gamma_j}$$

where l_j, c_j, and s_j denote luminance, contrast, and structure comparison functions at scale j, and on single-scale they are defined as:

$$\begin{aligned} l_j(x,y) &= \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1}, \\ c_j(x,y) &= \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}, \\ s_j(x,y) &= \frac{\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3}, \end{aligned}$$

Where μ_x,μ_y are local means, σ_x,σ_y are local standard deviations, and σ_{xy} is the local covariance and C_1, C_2, C_3 are stabilizing constants defined as:

$$C_1 = (K_1 L)^2, \qquad C_2 = (K_2 L)^2, \qquad C_3 = \frac{C_2}{2}$$

where L denotes the dynamic range of pixel intensities and K_1 and K_2, for 8-bit images $L = 255$, $K_1 = 0.01$, $K_2 = 0.03$.

In MS-SSIM, luminance(α_M) is applied only at the largest scale M, while contrast and structure are aggregated across all of them. For $M = 5$ scales, commonly used weights are

$$\beta_j = \gamma_j = \{0.0448, 0.2856, 0.3001, 0.2363, 0.1333\}$$

MS-SSIM values range from 0 to 1, with higher values indicating greater perceptual similarity. Because it evaluates the overall structure and content, the value tends to be higher than that of SSIM.

SSIM, ESSIM, and MS-SSIM metrics evaluations were implemented in Java. This is available in the codebase, path: src/main/java/org/example/tests/SSIM.java. The primary method is findSSIM, which takes two arguments (testPath and originalPath, both String) and outputs the SSIM, ESSIM, and MS-SSIM results to the console. For ESSIM, a sliding window with a stride of 1 is used for 8x8

blocks. For constants both of the variants use C_1, and C_2 the standard 8-bit color ($L = 255$) values were chosen, $K_1 = 0.01$, $K_2 = 0.03$.

4.3 Video Multi-method Assessment Fusion (VMAF)

VMAF is a modern perceptual video quality metric developed by Netflix to further improve the estimation of human subjective quality beyond the previously discussed metrics, PSNR and SSIM. Unlike systematic single-formula approaches to PSNR or SSIM, VMAF uses a unified predictive model trained on large datasets.

VMAF is a machine-learning model trained on human opinion scores for video datasets, producing a single quality metric that closely predicts how real viewers would perceive the video. A range of video features is considered, including blockiness, artifacts, texture changes, edge naturalness, and motion features.

Since VMAF is not readily implemented, the evaluation will use the ffmpeg version. The basic $vmaf.ps$ script (resources/vmaf.ps) was created; it takes two arguments (originalPath, testPath) and outputs the VMAF data back to the console. Version used - $vmaf_v0.6.1$. Example usage - `.\vmaf .ps1 birds.mp4 ABCABCbirds.mp4`.

4.4 Results

All presented metrics were obtained from various videos with different watermark strengths. In total three scenes will be tested, stills from them are visible in Figure 17. This is to verify that the watermark is not noticeable and stays invisible, as this is one of the requirements.

(a) *street.mp4* ***(b)*** *birds.mp4* ***(c)*** *vSquirrelEating.mp4*

Figure 17. *Visual impact analysis evaluation.*

All videos, and only their components, are available in the ”Visual analysis” folder. Within this folder, there are additional folders for each video and additional folders for each strength experiment. Additionally, the settings are the same across all videos and strengths. These include Regions: 6; Components: 3; Batch: 2; Recorded frames: 600; Color 1 (Green). Since we have 600 frames, the results are averaged across all frames. The unique variant evaluated is the same: A-B-C-A-B-C, to achieve a balanced, typical configuration.

4.4.1 Natural Motion Scene (file: street.mp4)

The scene depicts a casual stroll through a sidewalk in an urban setting. There is some camera motion and subject movement, such as a cyclist crossing a road. There are some smooth areas, such as the sky. On the sides, terraced buildings and tall signposts are visible. The camera is not static; numerous elements change.

Table 10 shows a clear trend: as watermark strength increases, all metrics decrease. This is entirely expected and partially verifies that the metrics are functioning as intended. For this scene, both strengths of 10 and 20 are appropriate, as all metrics remain within the limits. Because this is a non-static scene, PSNR degrades more rapidly because even minor spatial distortions are amplified across many changing pixels frame-to-frame. SIIM/MS-SSIM stays relatively high because they’re frame-based structure metrics and don’t fully capture temporal artifacts. Overall, with a strength of 10, the watermark remains practically invisible and would go unnoticed by the human eye.

Table 10. *Street video, visual impact metric results.*

Strength	PSNR-based metrics [dB]			SSIM-based metrics			VMAF
	PSNR	PSNR-HVS	PSNR-HVS-M	SSIM	ESSIM	MS-SSIM	
10	31.10	24.11	34.03	0.99	0.91	0.99	93.49
20	26.91	19.28	24.80	0.97	0.82	0.98	82.83
30	24.37	16.64	20.89	0.94	0.74	0.96	72.30
40	22.79	15.07	18.74	0.92	0.68	0.95	63.76
50	21.61	13.90	17.24	0.89	0.62	0.93	56.34

4.4.2 Static Fast-Motion, Heavily Textured Scene (file: birds.mp4)

The test video depicts a forest floor and a tree trunk with a bird-feeding setup attached. The camera is entirely static, but the scene contains dense natural textures (bark, needles, moss, foliage) and rapid local motion from many small birds flying around feeding. This scene is helpful because the background is stable and highly textured. Some watermarking techniques get stressed under very high textures.

The high texturing makes the watermark stand out; this is clearly reflected in Table 11. Even with the highest evaluated strength, these metrics remain within ranges for remaining visually imperceptible. SSIM metrics become highly saturated and typically don’t reflect subtle degradations well. ESSIM drops more rapidly than others is a clear clue: it’s more sensitive to edges/structure,

so it detects more damage. In the static but highly textured birds recording, increasing watermark strength causes a minor degradation in perceptual quality.

Table 11. *Birds video, visual impact metric results.*

Strength	PSNR-based metrics [dB]			SSIM-based metrics			VMAF
	PSNR	PSNR-HVS	PSNR-HVS-M	SSIM	ESSIM	MS-SSIM	
10	36.89	34.92	55.03	1.00	0.97	1.00	91.06
20	36.08	31.82	45.59	1.00	0.95	0.99	89.26
30	34.85	28.84	38.78	0.99	0.92	0.99	87.02
40	33.67	26.68	34.55	0.99	0.89	0.99	84.85
50	32.50	24.94	31.50	0.99	0.86	0.99	82.57

4.4.3 Low Motion Scene (file: vSquirrelEating.mp4)

The test video depicts a squirrel eating a nut in a natural outdoor environment. The camera is entirely static; minimal motion is present, primarily from the squirrel and the surrounding background. This scene provides a controlled, static-camera environment with minimal motion. Because the background does not move, any distortion introduced by the watermark should be easier to isolate and evaluate. For example, squirrel's fur and fine glass blades are high-frequency textures that are sensitive to minor changes in the frequency domain.

Based on Table 12, the results at 10 strength are minimal, with PSNR 36.05 dB and VMAF 92.49, and quality remains high. As strengths increase, visual degradation becomes apparent. Overall, these results indicate a practical quality-strength tradeoff where strength 10-20 preserves near-transparent quality, strength becomes borderline, and at 40-50 some clearly visible distortion becomes visible.

Table 12. *Squirrel eating video, visual impact metric results.*

Strength	PSNR-based metrics [dB]			SSIM-based metrics			VMAF
	PSNR	PSNR-HVS	PSNR-HVS-M	SSIM	ESSIM	MS-SSIM	
10	36.05	31.88	50.53	1.00	0.97	1.00	92.49
20	33.43	27.05	36.95	0.99	0.94	0.99	87.84
30	30.94	23.70	30.79	0.99	0.89	0.99	82.72
40	28.95	21.34	27.06	0.98	0.84	0.98	77.63
50	27.38	19.58	24.48	0.97	0.79	0.98	72.58

5 Robustness against attacks

This chapter does not aim to provide an exhaustive robustness evaluation against all possible attacks. As no new work was presented focused on robustness or resilience to attacks, this paper does not add much. Instead, the primary focus is on the proposed model, and this is merely a simple verification that the model did not significantly alter the FFT watermark robustness.

To demonstrate one of the key proposed models, being able to recover the watermark from a single frame. Tests were conducted on a single frame from the video or a screenshot; a single frame was captured and subjected to various tests. The robustness behavior summarized in Table 13 is characteristic of transform-domain watermarking based on the Fourier transform. The watermark exhibits strong resilience to photometric distortions (brightness and gamma changes), moderate noise, chroma degradation, and block-based quantization artifacts, all of which mainly affect spatial-domain representations while preserving dominant frequency components.

Operations that significantly alter the spatial frequency, such as strong blurring, aggressive downscaling, and combined attacks, can degrade the watermark to the point that it is not extractable. These behaviors are expected in FFT-based schemes, as these attacks suppress or redistribute the frequency bands that were used for embedding.

Table 13. *Image-domain robustness attacks, survivability results, and corresponding test images.*

Attack	Watermark survived	Image filename
JPEG compression (Q=70)	✓	`jpeg_q70.png`
JPEG compression (Q=30)	×	`jpeg_q30.png`
Gaussian noise ($\sigma = 10$)	✓	`gauss_noise_10.png`
Gaussian noise ($\sigma = 25$)	×	`gauss_noise_25.png`
Salt-and-pepper noise (1%)	✓	`saltpep_1%.png`
Gaussian blur ($k = 7$)	×	`blur_k7.png`
Motion blur ($k = 15$)	✓	`motion_k15.png`
Rescaling ($0.5\times$)	×	`resize_0.5.png`
Rotation (5°)	✓	`rotate_5deg.png`
Center crop (90%)	✓	`crop_0.9.png`
Brightness/contrast ($+20$, $1.1\times$)	✓	`b+20_c1.1.png`
Gamma correction ($\gamma = 1.4$)	✓	`gamma_1.4.png`
Perspective warp	✓	`persp_0.02.png`
Cutout (15%)	✓	`cutout_0.15.png`
Block quantization (8×8)	✓	`block_q8_s20.png`
Chroma blur ($k = 9$)	✓	`chroma_blur_k9.png`
Combined mild attack	×	`combined_mild.png`

All of the images of how attack looked, and what was extracted is present in folder ”Robustness test” alongside Jupiter notebook file used to create those attacks.

6 Results

The following requirements has been raised for the system:

Core functional requirements

- **Payload:** The watermark must carry enough information to identify the viewer.
- **Robustness to transformations:** The watermark should tolerate mild compression, format changes, scaling, cropping, frame manipulations, drop/insert/swap.
- **Recoverability:** The watermark should be recoverable from a short recording, even under adverse conditions, with only one frame.

Non-functional requirements

- **Scalability:** The system should scale reasonably (no worse than linear time complexity) as the number of viewers increases.
- **Codec and format:** The model should remain effective across H.264/AVC, HEVC, and other state-of-the-art codecs. It should be versatile and generalized, without being tied to a specific codec.
- **Real-time:** The system should be capable of watermarking in real-time.

Fulfilling these requirements is the primary goal. Next we review how each requirement is met.

Payload

This refers to the amount of information a watermark can carry, as the proposed system relies on a single frame for extraction. The payload is denoted by N, the number of supported viewers or unique variants. The exact practical limit is not explored, but realistically, at least millions is within the range, with $R = 13$. The millions refers to natural numbers; for example, with $R = 13$ and $N = 3^{13} = 1\,594\,323$, not bits or bytes. The theory is based on Theorem 3.2.2, as this theorem underpins scalability and the required payload. The actual appearance of the payload is shown in Figure 15.

Robustness to transformations

This is partially satisfied by choosing FFT-based, frequency-domain watermarking. Minor robustness verification is presented in Table 13. Regarding frame manipulations - drop/insert/swap, as a full watermark payload exists in a single frame, these attacks don't matter. Cropping depends on the regions we select; overall, it is distributed across the frame.

Recoverability

Single-frame recovery is by the model's design. This is vital, as a single frame sometimes can be sufficient to cause severe damage to media authors or security and military-related applications. Recovery across multiple frames is also recommended to increase confidence in the extracted sequence. This is supported by Figure 15, as the extraction was performed on a single frame.

Scalability

There is much to unpack here, since many components of the system should reasonably scale. To begin with, the most costly elements of video broadcasting are encoding and storage, with encoding typically accounting for the majority of costs. The total encoding required for a fixed frame size to support at least N viewers is constant, as shown in Lemma 3.2.6; the same holds for storage. This is the best-case scenario in terms of complexity. Still, we can not state that it is the overall best possible, as the constant factor in Big O is hidden, and all constant-time complexities are not equal. The most significant victory here is that encoding/storage scaling is not directly linked to N.

Another significant aspect is watermarking, the creation of frames for encoding. As expected, this should also be scalable. The watermarking pipeline is based on FFT, which itself inherently starts with complexity of $O(P \log P)$, and introducing N variants under naïve approach, it would become $O(N \cdot P \log P)$, which is linear scaling regarding N. This is poor scaling; to improve it, a scaling structure was proposed, whose theoretical foundation is presented and proven in Theorem 3.2.2. As the theorem only proposes the structure, it does not explore the complexity of the pipeline itself. This is done in Theorem 3.2.10, which reveals that the pipeline has a complexity of $\Theta(P \log P \cdot \log N)$, showing that N increases logarithmically and not linearly like a naive approach would. $\log N$ scaling is usually the best achievable for complex systems and is highly sought after because the next faster complexity class is constant time, $O(1)$. Sometimes theoretical assumptions do not closely align with practice, which is why two experiments were conducted to provide practical validation for Theorem 3.2.10. Complexity with respect to N is firmly validated by practical run-times, as shown in Figure 7. The experiment considered how runtime grows from $N = 3$ to $N = 3.4 \cdot 10^9$; time does not increase linearly but follows a logarithmic trend. Regarding scaling with respect to P, this was also verified similarly Figure 9, but nothing new was discovered, as the complexity of the FFT is well known and has been researched previously.

Codec and format

This is fully satisfied by choosing an FFT-based approach. Watermark insertion does not occur during the encoding process. This might appear as an unusual requirement, but some watermarking techniques insert watermarks during the encoding process. This leaves them vulnerable to arguably the most common attack: ”accidental” compression and/or format/codec change. In practical deployment scenarios, media content is rarely preserved in its original form. Uploading videos or images to online platforms triggers automatic re-encoding, resolution adjustment, chroma subsampling, bit-rate normalization, compression, and other processes.

Real-time

This is a highly abstract requirement that can be interpreted in different ways. To clarify the context, practical viability refers to the result’s viability under realistic hardware conditions for the problem at hand. This was evaluated in Chapter 3.3. It analyzes a 1080p, 60 fps color video; this setup remains relatively standard in the streaming industry. Because times depend on N, the supported viewer count, multiple counts were tried out. Based on the results, $R = 5$ regions, or $N = 3^5 = 243$ viewers, can be readily supported by the current software and hardware setup. This is a highly bottlenecked variant, as CPU-based encoding is suboptimal. For regions 8 and 10, with $R = 10$, $N = 3^{10} = 59\,049$ is readily achieved with an unlocked GPU, since the encoding time would be

much lower. There are also numerous software optimizations available to improve this further. This is a proof-of-concept to evaluate the viability of using this model in real-time applications. Overall results should be interpreted as strong evidence of the model's high potential for real-time application on commercial hardware. This is speculative, but I firmly believe that multiple 4K-resolution videos can be watermarked into millions of variants in real time with commercial-grade hardware and appropriate software.

Real-time application would likely be a small part of this model's use, as most paid, walled, or confidential/secret media content is never delivered live. This model can also be easily applied to those use cases. A real-time application is a significant milestone to achieve, in order to have additional flexibility.

Competitive model

Comparing this model with the A/B watermarking proposed by ETSI reveals the main diffrence. First, the time threshold for identification is reduced; the proposed model can identify the watermark solely from a screenshot. In A/B watermarking, extraction requires a lengthy copy: "As the unique ID is obtained ... the acquired content shall be of several minutes"[5], meaning from a few-second copy, identification is impossible. This is true even under perfect conditions (no compression, noise, etc.). In the best case, A/B watermarking would only recover a small portion of the watermark, which does not narrow down the leaker at all. The proposed model is designed to address use cases beyond A/B watermarking. No other models are comparable to the presented one.

7 Conclusions

A vital gap in the research was discovered. There are no practical solutions that enable us to identify content-leak sources in short-duration recordings, particularly single-frame leaks.

The strict requirements create a unique system that offers several desirable properties, such as scalability, robustness, single-frame extraction, and real-time performance.

It achieved logarithmic scaling with respect to N, the number of supported viewers. The watermarking pipeline achieves a final complexity of $\Theta(P \log P \cdot \log N)$. At the same time, it preserves frequency-domain robustness and enables identification from a single frame. Additionally, achieving constant computational complexity $\Theta(1)$ for encoding and storage.

Practical implementation demonstrates that system scaling follows the theoretical work presented and meets the system requirements. Visual analysis confirms that watermarks remain hidden and imperceptible to the human eye.

8 Future research

Further theoretical improvements should be explored to reduce the watermarking pipeline complexity. This model, for example, found a practical application for IFFT pruning, as prior to this pruning IFFT had limited to no practical use.

FFT is just one of the possible frequency-domain techniques; other ones should be considered. There are hundreds of wavelet functions, most of which are potentially implementable and align with the proposed model. FFT is likely not the best approach; therefore, other techniques should be explored to implement the same scalable model.

There is no visual analysis metric for evaluating transitions between regions. In some use cases, the transition may be highly visible and missed by standard metrics.

Watermark pattern selection should be explored. What images are best suited, have the best survival rate while staying visually hidden. A dynamic watermark strength selection should also be explored, since textured areas showed better capabilities to hold stronger watermarks.

More in-depth visual impact and robustness evaluation should be done. With a focus on getting better results and building a better understanding of how watermark pattern choice or strength works. Maybe different methods are superior for watermark insertion.

References and sources

[1] K. An, Z.-M. Lu, X.-C. Sun, Z.-H. Wang. “Multipurpose Video Watermarking Algorithm for Copyright Protection and Tamper Detection.” In: *Multimedia Tools and Applications* 83 (2024), pages 51647–51668. https://doi.org/10.1007/s11042-023-17558-1. (Viewed 2025-03-12).

[2] Art of Problem Solving Wiki. *Proofs of AM-GM*. Art of Problem Solving. URL: https://artofproblemsolving.com/wiki/index.php/Proofs_of_AM-GM (viewed 2025-09-06).

[3] W. Chen, Y. Li, Z. Niu, Y. Xu, A. Keskinarkaus, T. Seppänen, X. Sun. “Real-time and screen-cam robust screen watermarking.” In: *Knowledge-Based Systems* 302 (2024), page 112380. https://doi.org/10.1016/j.knosys.2024.112380. (Viewed 2025-03-12).

[4] K. Egiazarian, J. Astola, N. Ponomarenko, V. Lukin, F. Battisti, M. Carli. “New full-reference quality metrics based on HVS.” In: *CD-ROM Proceedings of the Second International Workshop on Video Processing and Quality Metrics for Consumer Electronics (VPQM-06)*. 2006, pages 1–4.

[5] European Telecommunications Standards Institute. *DASH-IF Forensic A/B Watermarking: An interoperable watermarking integration schema*. ETSI TS 104 002 V1.1.1. European Telecommunications Standards Institute (ETSI), 2023. URL: https://www.etsi.org/deliver/etsi_ts/104000_104099/104002/01.01.01_60/ts_104002v010101p.pdf (viewed 2025-05-08).

[6] J.-U. Hou. “MPEG and DA-AD Resilient DCT-Based Vid‘eo Watermarking Using Adaptive Frame Selection.” In: *Electronics* 10.20 (2021), page 2467. https://doi.org/10.3390/electronics10202467. (Viewed 2025-03-12).

[7] X. Yang, Z. Zhang, Y. Jiao, Z. Li. “A Robust Video Watermarking Algorithm Based on Two-Dimensional Discrete Fourier Transform.” In: *Electronics* 12 (2023), page 3271. https://doi.org/10.3390/electronics12153271. (Viewed 2025-03-12).

[8] G. Lin, W. Luo, P. Zheng, J. Huang. “An Audio Watermarking Method Against Re-Recording Distortions.” In: *Pattern Recognition* 162 (2025), page 111366. https://doi.org/10.1016/j.patcog.2025.111366. (Viewed 2025-03-12).

[9] A. Malanowska, W. Mazurczyk, T. K. Araghi, D. Megías, M. Kuribayashi. “Digital Watermarking—A Meta-Survey and Techniques for Fake News Detection.” In: *IEEE Access* 12 (2024), pages 36311–36338. https://doi.org/10.1109/ACCESS.2024.3374201. (Viewed 2025-03-12).

[10] N. Ponomarenko, F. Silvestri, K. Egiazarian, M. Carli, J. Astola, V. Lukin. “On between-coefficient contrast masking of DCT basis functions.” In: *CD-ROM Proceedings of the Third International Workshop on Video Processing and Quality Metrics for Consumer Electronics (VPQM-07)*. 2007, pages 1–4.

[11] S. Rana. “3D Video Watermarking for MVD Based View-Synthesis and RST Attack.” In: *Multimedia Tools and Applications* 83 (2024), pages 26775–26795. https://doi.org/10.1007/s11042-023-16481-9. (Viewed 2025-03-12).

[12] W. El-Shafai, M. A. Fouda, E.-S. M. El-Rabaie, N. A. El-Salam. “A Comprehensive Taxonomy on Multimedia Video Forgery Detection Techniques: Challenges and Novel Trends.” In: *Multimedia Tools and Applications* 83 (2024), pages 4241–4307. https://doi.org/10.1007/s11042-023-15609-1. (Viewed 2025-03-12).

[13] B. Singh, G. Kasana. “A Review of Digital Watermarking Techniques: Current Trends, Challenges and Opportunities.” In: *Web Intelligence* 22 (2024), pages 523–553. https://doi.org/10.3233/WEB-230280. (Viewed 2025-03-12).

[14] Y. Sun, G. Srivastava. “Digital Watermarks for Videos Based on a Locality-Sensitive Hashing Algorithm.” In: *Mobile Networks and Applications* 28 (2023), pages 1724–1737. https://doi.org/10.1007/s11036-023-02240-5. (Viewed 2025-03-12).

[15] Z. Wang, E. Simoncelli, A. Bovik. “Multiscale structural similarity for image quality assessment.” In: *The Thrity-Seventh Asilomar Conference on Signals, Systems & Computers, 2003*. Volume 2. 2003, 1398–1402 Vol.2. https://doi.org/10.1109/ACSSC.2003.1292216.

[16] D. Zolba. “Efektyvus vandens ženklų taikymas dideliam žiūrovų skaičiui identifikuoti.” In: *Lietuvos matematikos rinkinys 66(B), 2025*. 2025, p.139–150. https://doi.org/10.15388/LMR.2025.44464.