



# Deepshield: Technical Report

IABG mbH, Innovation Center

Beauftragung zur IABG Angebotsnummer IZ10\_24046601\_V01  
vom 26.11.2024

IABG Industrieanlagen-Betriebsgesellschaft mbH  
Einsteinstrasse 20, 85521 Ottobrunn, Germany, Telefon: 089 6088-0




## Changes Overview / Document Properties

### Modification history

Version	Datum	Änderungen	Autor	Status
0.1	03.24.2025	Definition of the document structure and attribution of content to chapters.	Njeh Mohamed Notaro Paolo	Draft
0.2	04.11.2025	Writing the content of the chapters.	Charrez Olivier Ilanchezian Indu Notaro Paolo Vaghasiya Nehal	Draft
0.3	04.22.2025	First version of the document.	Charrez Olivier Njeh Mohamed Notaro Paolo	Draft
0.4	04.25.2025	Review	Riedl Martin Dr. Notaro Paolo	Final

### Document properties

Title of the document	Technical Report IABG Deepshield
Type of the document	Technical Report
Version	1.0
Responsible persons	Dr. Martin Riedl (IZ10) Mohamed Njeh (IZ10)
Date	04.25.2025
Signatures	

## Summary

This report presents the IABG contribution to the Deepshield project, aimed at tackling the spread of deceptive multimedia content. The system employs a three-fold approach to ensure authenticity and semantic integrity: a) robust watermarking, utilizing frequency-domain techniques for imperceptible and resilient marking; b) Trusted Execution Environments (TEEs), leveraging secure hardware isolation for protected processing; and c) semantic integrity analysis, incorporating advanced techniques to detect contextual manipulations. Through this multi-layered methodology, Deepshield enhances the trustworthiness of digital media, mitigating the risks associated with deepfakes and disinformation. The report delves into the functional components, implementation challenges, and evaluation outcomes of the project.

## Table of Contents

<b>Changes Overview / Document Properties .....</b>	<b>2</b>
<b>Summary .....</b>	<b>3</b>
<b>Table of Contents .....</b>	<b>4</b>
<b>1 Introduction .....</b>	<b>5</b>
<b>2 Related Work .....</b>	<b>8</b>
2.1 Comprehensive solutions for image forgery detection .....	8
2.2 Watermarking Approaches .....	10
2.3 Semantic Integrity Approaches .....	11
2.4 Trusted Execution Environments .....	12
<b>3 Approach .....</b>	<b>14</b>
3.1 Overall System Architecture .....	14
3.2 Watermarking System .....	16
3.3 Trusted Execution Environment (TEE) .....	21
3.4 Multi-layered Semantic Integrity Verification .....	24
<b>4 Evaluation .....</b>	<b>30</b>
4.1 Experimental Setup .....	30
4.2 Watermarking Evaluation Results .....	31
4.3 Semantic Integrity Evaluation Results .....	36
4.4 Overall Summary and Takeaways .....	39
<b>5 Challenges .....</b>	<b>40</b>
5.1 Watermarking .....	40
5.2 Trusted Execution Environment .....	42
5.3 Semantic Integrity .....	43
<b>6 Conclusion .....</b>	<b>45</b>
6.1 Possible Extension and Future Work .....	45
<b>7 References .....</b>	<b>47</b>

# 1 Introduction

The widespread propagation of visual content online has increased concerns surrounding its authenticity and the safeguard of intellectual property protection. The rapid advancement of Generative Artificial Intelligence (GenAI) has intensified this issue, enabling the creation of highly realistic deepfakes, i.e., synthetic media that imitate real people and events with alarming precision [1], [2]. Moreover, authentic images can be deliberately cropped, re-framed, or paired with misleading narratives, without altering the image pixels themselves.

These AI-generated or modified images and videos have been increasingly exploited in misinformation and disinformation campaigns, thereby distorting public perception and undermining trust in digital media and posing serious threats to individuals, organizations, and democratic institutions [3]. The increasing number and the ease of accessibility of tools that enable media manipulation further amplifies these risks, resulting in a serious and urgent challenge in the current digital landscape.

Addressing the deepfake phenomenon is essential to preserving the integrity of digital information. Without robust detection and verification mechanisms, malicious use of deepfakes can result in identity fraud, political manipulation, and the erosion of public trust in media and official communication channels [4]. As emphasized by organizations such as the European Commission and UNESCO, unchecked disinformation campaigns threaten social cohesion, democratic processes, and global security [5]. The ease with which manipulated images can be created and disseminated necessitates urgent countermeasures that are scalable, efficient, and adaptive to an evolving threat landscape.

Several methods have emerged from industry and academia to combat the proliferation of deepfakes, most notably:

- Sensity AI [6] provides an enterprise-grade platform for detecting manipulated media, leveraging deep learning models that analyze content artifacts (e.g., inconsistent lighting, facial asymmetries) and metadata. It claims detection accuracies of 98% across formats.
- Intel's FakeCatcher [7] uses Photoplethysmography (PPG) to analyze subtle blood flow patterns in video pixels, distinguishing genuine videos from AI-generated ones in real-time with a reported accuracy of 96%.
- Reality Defender [8] focuses on real-time deepfake detection, providing browser-integrated tools and video call alerts to flag synthetic content, targeting fraud prevention in high-stakes scenarios.

Despite their utility, current solutions exhibit several limitations. These systems generally employ machine learning classifiers, facial landmark tracking, or temporal consistency analysis to detect anomalies. However, many require continuous model updates to remain

effective, a significant drawback in the context of rapidly evolving generative and adversarial learning techniques.

Other important limitations include:

- **Lack of Traceability:** Most solutions detect manipulations but cannot verify the original source of the image. They fail to distinguish between benign and malicious edits, as they lack access to or provenance of the original content.
- **Compromised Semantic Integrity:** Images may undergo little to no alteration yet still convey false narratives due to small presentation distortions, such as selective cropping, warping, color modification, misleading captions, or partial display, which evade detection by conventional image analysis methods.

In contrast to that, we propose a solution that does not only focus on detecting deep fakes, but also aims to identify and verify content that is both authentic and preserves its original semantics.

The rationale behind this approach lies in our core assumption: the value of information is intrinsically linked to its trustworthiness. Trust, in turn, depends on identifiable entities that produce authentic content while maintaining its original meaning. Therefore, rather than merely filtering out deep fakes, our objective is to expand the space of trustworthy information available to society especially in contexts where such information is critical for opinion formation and decision-making.

A comprehensive solution must utilize a multi-layered system of protection, which includes cryptographic signatures, trustworthy hardware environments, and a distributed ledger system for end-to-end provenance and integrity assurance.

To address these challenges, we introduce a multi-layered watermarking and verification pipeline, as part of the Deepshield Project. Our solution combines:

- Cryptographic watermarking techniques that enable verification of content origin, which are shown to be robust against a wide set of intentional and stochastic perturbations, with a controllable level of perceptibility.
- State-of-the-art machine learning models, such as Visual Large Language Models (VLLMs) for advanced semantic and object-level integrity checks, identifying not just manipulations but also shifts in contextual meaning. These algorithms are proven to be adaptive so to minimize the need for frequent retraining.
- Trusted Execution Environments (TEEs) for edge devices, which protect sensitive data used to produce a valid watermark and actual related computations within a secluded environment, minimizing exposure to potential vulnerabilities.

This integrated design enables detection of both technical alterations and semantic misuse, such as misrepresentation of context or narrative framing, i.e. threats not addressed by

image analysis alone. Our pipeline offers improved traceability, tamper detection, and semantic robustness, providing a more holistic approach to content verification.

Our approach is designed to overcome the scalability, generalizability, and trust limitations of current tools. By employing a hybrid model that combines cryptographic verification within secure hardware environments with advanced AI-based analysis, we achieve:

- Semantic awareness, distinguishing not only if an image has changed, but whether its meaning has shifted.
- Resilience against curious adversaries on the hardware level that might want to get access to the original non-watermarked image, or the private key used for the generation of the watermark. This is enabled by execution within isolated and verifiable environments (TEEs).
- Readiness for integration with other approaches and solutions to extend the approach further. (e.g. Watermark and register known deepfakes and improve the detection workflow)

This positions our solution as a trustworthy, privacy-preserving, and robust alternative to existing deepfake detection systems. Additionally, its modular layered design allows select existing solutions to be integrated as complementary sub-tools when advantageous.

The evaluations presented in this report demonstrate promising performance of our system. Our watermarking detection method achieves excellent robustness ( $NCC > 90$ ) and watermark detection ( $CATSO > 84.18$ ) across a wide set of distortions and low watermarking strength ( $\alpha = 0.05$ ), indicating strong imperceptibility and resilience. Additionally, our semantic analysis module effectively flags semantic and contextual alterations (accuracy  $> 0.77$ ,  $F1 > 0.72$ ), even when the visual appearance remains unmodified.

## 2 Related Work

In this section, we review prior work across four key areas that intersect with our proposed approach: (1) comprehensive solutions for image forgery detection, (2) digital watermarking techniques, (3) methods for assessing semantic integrity, and (4) the application of Trusted Execution Environments (TEEs) to ensure confidentiality during the runtime of sensitive operations, such as private key handling in watermark generation and insertion.

### 2.1 Comprehensive solutions for image forgery detection

Table 2 summarizes some existing image authentication and deepfake recognition systems from industry and academia.

Table 2: Overview of existing image authentication and deepfake recognition systems.

Software	AI Models	Cryptographic Signatures	Trusted Execution Environment (TEE)	Other Technologies
Nguyen et al. [1]	Multi-task CNNs (segmentation + classification)	–	–	Facial forgery localization and detection
Sensity AI [6]	GAN-based detection	–	–	Real-time monitoring, pixel-level analysis
FakeCatcher [7]	Landmark detection, OpenVINO	–	– (May use Intel SGX in the future)	PPG, physiological signal analysis
Reality Defender [8]	Proprietary ML models for real-time detection	–	–	Real-time alerts during video calls, browser integrations
HyperVerge [11]	Deep learning algorithms	–	–	–
Sentinel AI [12]	CNN, deep learning, neural networks	–	–	Temporal analysis, flicker detection
Resemble AI [13]	Deep neural networks	PerTh (Neural watermarks)	–	Audio fingerprinting
DeepBrain AI [14]	CNN, deep learning	–	–	Watermarking, lens distortion analysis



<b>DeepShield</b>	CNN, VLLMs	Cryptographic signatures, Water-marking	ARM TrustZone (through OPTee)	Blockchain, semantic analysis
-------------------	------------	---	-------------------------------	-------------------------------

Nguyen et al. [1] propose a multi-task deep learning architecture for detecting and localizing image manipulations, particularly facial forgeries. While effective at pixel-level tampering detection, their approach lacks cryptographic traceability and does not address semantic drift or image provenance, which are key components of our system.

Sensity AI [6] delivers an enterprise-level platform using GAN-based analysis verifying content authenticity. While it supports real-time monitoring and media fingerprinting, it lacks semantic awareness and secure execution environments, which our system includes to ensure both perceptual and contextual robustness.

Intel FakeCatcher [7] detects deepfakes in real time by analyzing physiological signals like skin blood flow, extracted from video frames. While it excels in detecting facial synthesis, FakeCatcher is modality-specific and lacks support for cryptographic validation or semantic reasoning, which our system provides across diverse image contexts.

Reality Defender [8] focuses on real-time detection of deepfakes, especially during high-risk video communications, and is designed for integration into web environments. However, it does not verify provenance or offer cryptographic watermarking, nor does it assess semantic shifts, which are capabilities that are central to our multi-layered defense.

HyperVerge [11] offers a commercial tool for real-time image and video forgery detection, optimized for speed and integration into digital onboarding pipelines. However, it operates purely at the content level and does not incorporate trusted execution or origin verification, making it less robust in adversarial or high-integrity contexts.

Sentinel [12] leverages CNNs and temporal consistency checks such as flicker detection and facial landmark shifts to identify deepfakes in video. Unlike our pipeline, Sentinel does not integrate cryptographic protections or contextual integrity verification, which limits its ability to detect subtle or malicious reinterpretation of unchanged content.

Resemble AI [13] developed PerTh, a neural audio watermarking scheme that embeds imperceptible data into synthesized speech, providing audio origin traceability. Our system expands upon this concept by integrating visual watermarking into image content and coupling it with secure semantic analysis and TEEs, addressing both media and meaning.

DeepBrain AI [14] uses CNN-based analysis and lens distortion detection to identify tampered visual content, often applied in avatar or video production pipelines. Our solution differs in that it adds cryptographic watermarking and secure semantic analysis to detect misuse even when no visual artifact is present.

In contrast to previous approaches, our system combines cryptographic techniques, Trusted Execution Environments (TEEs), and Visual Language Models (VLLMs) to provide multi-layered detection of visual tampering and contextual manipulation. Unlike other systems, it verifies both the origin and meaning of an image within a secure pipeline, enabling both provenance and semantic integrity checks.

## 2.2 Watermarking Approaches

Watermarking is a well-established method for asserting content authenticity and detecting manipulation in digital media. While a few simple watermarking methods apply modifications directly in the pixel space, most of the traditional approaches operate in the frequency domain, where watermark embedding is performed in transformed representations of the image such as the Discrete Cosine Transform (DCT) and Discrete Wavelet Transform (DWT).

Rezaei and Haidar [15] introduce a blind watermarking scheme using adaptive embedding strengths and secret keys, achieving imperceptibility and basic robustness. Similarly, DCT-domain watermarking methods [16] embed encrypted messages into low-frequency coefficients using techniques like Horizontal/Vertical Parity Mapping (HPM/VPM), though they remain vulnerable to geometric attacks such as cropping and rotation.

To improve robustness, hybrid frequency-domain techniques [17] combining 2DWT and DCT have been proposed. These embed watermark data into the most stable low-frequency bands of multi-level wavelet-decomposed images, producing watermarking schemes that maintain a peak signal-to-noise ratio (PSNR)  $\geq 40$  dB, structural similarity index (SSIM)  $\geq 0.9$ , and bit correction ratio (BCR)  $\geq 95\%$  under various attack types including filtering, histogram equalization, and lossy compression.

More recent work has shifted toward deep learning-based watermarking systems. The approach described in [18] integrates interest point detection with deep learning models to improve watermark placement and extraction under geometric transformations. Other schemes combine watermarking with cryptographic security [19], generating pseudo-random keys to determine embedding positions and encrypt the payload, enhancing both robustness and resistance to reverse-engineering.

Neural approaches like FreqMark [20] utilize variational autoencoders (VAEs) to embed multi-bit watermarks that demonstrate high bit accuracy ( $>90\%$ ) under diverse transformation attacks. Similarly, Watermark Anything [21] demonstrates robust watermark recovery from small image patches using neural encoders and decoders, even on low-resolution images. However, these systems often rely on large models and opaque decision mechanisms. To support fair evaluation of such techniques, the WAVES framework [22] proposes a standardized benchmarking suite to assess watermark robustness. Despite these

advances, emerging work [23] highlights a critical vulnerability: diffusion models like Stable Diffusion can act as effective watermark removal mechanisms, rendering many deep or classical watermarking techniques fragile under modern generative attacks.

While deep learning-based methods offer promising levels of robustness and adaptability, our system opts for traditional frequency-domain watermarking techniques. This decision is primarily driven by the difficulty of executing neural watermarking pipelines within Trusted Execution Environments (TEEs). Neural models require substantial computational resources, have larger memory footprints, and introduce attack surfaces that complicate secure deployment. Moreover, the absence of a formal authenticity proof via watermarking renders the content inherently untrustworthy, making any reliance on such unverifiable data risky. In contrast, traditional DWT-DCT watermarking methods can be efficiently implemented in constrained environments with predictable execution costs and formalized robustness guarantees, making them more compatible with secure and lightweight architectures.

## 2.3 Semantic Integrity Approaches

Assessing semantic integrity in images goes beyond detecting low-level pixel alterations, aiming instead to capture changes that affect the meaning, message, or context of visual content. Several complementary strategies have been proposed to identify such manipulations.

Image Captioning and Embedding Similarity approaches generate captions using vision-language models (VLMs) such as CLIP or BLIP and convert these captions into embeddings using models like Sentence-BERT. Related works [24], [25], [26] detect semantic drift by comparing embeddings between different versions of the same image, flagging significant shifts in content description. These methods are particularly effective for identifying manipulations like cropping, object removal, or altered background elements that preserve visual realism but change the perceived meaning. However, their performance is limited by the quality of the captioning models and their susceptibility to bias in language generation.

Object Detection and Scene Comparison techniques rely on models like YOLO, Faster R-CNN, or DETR to extract objects and spatial relationships from images. By comparing the object sets or generating scene graphs [27], changes in structure or relationships (e.g., “person holding a flag” vs. “person standing”) can be detected. Methods like SuperGlue and LoFTR further improve correspondence matching. These approaches are well-suited for detecting localized changes, but they struggle with subtle context shifts that don't affect scene structure.

Image Forensics and Metadata Analysis involves extracting EXIF metadata or detecting pixel-level tampering using methods such as Error Level Analysis (ELA). These techniques are well-established for flagging authenticity violations [28] but are ineffective against manipulations that preserve low-level image statistics while altering content meaning.

Direct Image-to-Image Comparison methods compare suspect and original images directly using keypoint matching (e.g., SIFT, ORB [50]) or deep similarity metrics. These can highlight localized edits effectively [29], but only when a trusted reference is available and cannot detect semantic reinterpretation like reframing or captioning changes.

Contextual Relationship Validation methods [30] incorporate external knowledge or use multi-modal reasoning to validate whether the content of an image makes sense. For example, knowledge graphs or visual question answering (VQA) models can be used to flag scenes that include impossible or unlikely combinations of objects. While powerful, these techniques are computationally expensive and often require domain-specific training.

Semantic Segmentation and Focus Analysis attempts to segment the image into semantically meaningful regions using models like DeepLab or Mask R-CNN [31], followed by attention comparison to assess changes in focus or composition. This can be effective for identifying changes in salience (e.g., who is emphasized in a news photo), but requires well-aligned segmentation models and fails to capture nuanced narrative changes.

Finally, Perceptual Hashing offers lightweight, tamper-sensitive fingerprinting (e.g., via pHash or dHash) to detect fine-grained image manipulation. While useful for flagging local retouching or contrast adjustments, this method cannot infer whether changes carry semantic weight.

To address the limitations of individual techniques, recent proposals [24, 44, 45] recommend combining multiple approaches for robust semantic verification, integrating low-level forensic methods with high-level captioning and reasoning.

In our work, we focus initially on captioning and object detection, as they offer a compelling tradeoff between coverage, interpretability, and scalability. Captioning captures abstract semantic meaning, while object detection anchors comparisons in concrete visual elements. Together, they enable detection of both narrative shifts and structural alterations, making them a pragmatic starting point for semantic integrity assessment.

## 2.4 Trusted Execution Environments

A Trusted Execution Environment (TEE) is a hardware-based isolated environment that ensures code and data confidentiality and integrity during execution. TEEs operate in parallel with the normal operating system, isolating sensitive operations in a secure area of the processor to defend against unauthorized access, tampering, or observation [32], [33].

Popular hardware implementations of TEEs include Intel Software Guard Extensions (SGX) and ARM TrustZone. ARM TrustZone provides hardware-level separation of a device into two execution environments: the Normal World (for general applications) and the Secure World (for sensitive applications). OP-TEE is an open-source operating system that runs in the Secure World, providing a runtime environment and API support for developing Trusted Applications (TAs) [34], [35]. The ARM Trusted Firmware initializes the Secure World during system boot, and all communication between the Normal and Secure Worlds is handled via a dedicated secure channel.

OP-TEE [36], [37] implements the GlobalPlatform TEE Internal Core API and Client API standards, enabling portable and modular TA development in C or Rust. While OP-TEE is well-supported on ARM-based platforms such as the Raspberry Pi 3, it can also be emulated on x86 machines using QEMU, which provides an ARM environment for prototyping and debugging [38], [39]. However, OP-TEE does not support Python-based Trusted Applications, making C or Rust the only viable languages for secure computation within this context [40].

Several research efforts have explored integrating deep learning workloads into TEEs. Examples include running lightweight DNN layers within ARM TrustZone [41], verifiable and privacy-preserving inference with Intel SGX [42], and secure aggregation of PyTorch models using PySyft [43]. However, these implementations remain limited in terms of scalability, model size, and performance overhead, especially in real-time applications or resource-constrained devices.

Given these constraints, our system prioritizes simplicity and robustness by deploying lightweight, C-based Trusted Applications within OP-TEE on ARM platforms. Unlike deep learning-based watermarking or detection schemes that demand high computational resources, our choice of conventional algorithms aims for efficient deployment within the limited execution and memory constraints of TEEs, while maintaining strong security guarantees.

### 3 Approach

In this chapter, we present our approach for secure multi-layered watermarking and verification. We first discuss our overall system architecture (3.1), composed of our three main components (watermarking, TEE, semantic integrity), then we describe each of the three components in detail (3.2, 3.3, 3.4).

#### 3.1 Overall System Architecture

Our full system architecture is depicted in the diagram below (Figure 1).

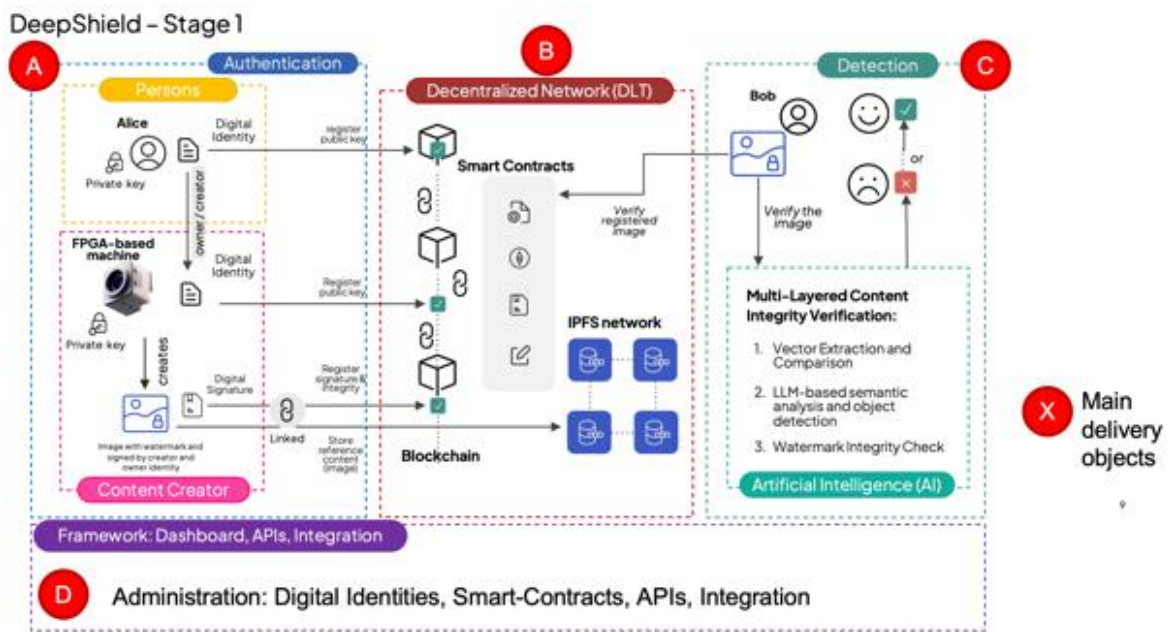


Figure 1: Overall Building Blocks of Deepshield and their general possible interactions

We define two categories of users: content creators and verifiers. Content creators (like Alice, red circle **A**) generate images on their local edge devices. These images are immediately watermarked upon creation, to ensure authenticity and track their origin (author). This process is secured through the use of private keys for signing and by executing the watermarking process on the Trusted Execution Environment (TEE) of the edge device. After their image has been watermarked, the content creator can voluntarily choose to register their content in the decentralized network (red circle **B**) which acts as a distributed ledger and guarantees transaction verification through smart contracts. In a second step, verifiers (like Bob, red circle **C**) may want to associate a new public content (candidate image) to a specific source for authenticity and copyright protection. For this purpose, they submit their candidate image for verification to the decentralized network, which acts as match system and associates the new image to one of the images registered previously by a content creator (thanks to the watermarks inserted during content creation). If such match

is found, the watermarked image and the corresponding author is returned. Moreover, in the presence of a match, the multi-layered semantic integrity verification (C) provides guarantees of semantic integrity, i.e. whether the candidate image does not diverge in meaning and/or interpretation the registered image.

### 3.1.1 Interfaces between System Components

For the development of the PoC, we have conceptualized a set of key interfaces designed to support image registration, verification, and interaction with blockchain and decentralized storage systems. These interfaces aim to facilitate smooth integration between the image processing pipeline and the underlying infrastructure including the blockchain.

- ***register\_image(watermarked\_image, watermark\_data)***
  - Purpose: Registers a watermarked image by storing the image and its associated metadata.
  - Parameters:
    - *watermarked\_image*: The image with an embedded watermark.
    - *watermark\_data*: The watermark matrix or metadata.
  - Returns: Confirmation of successful registration or error details.
- ***get\_evaluation\_requests(auth\_token=None)***
  - Purpose: Fetches a list of pending evaluation requests requiring verification.
  - Parameters:
    - *auth\_token* (optional): Token for authentication and access control.
  - Returns: A list of evaluation request objects, each containing relevant metadata and status information.
- ***update\_evaluation\_request(new\_status)***
  - Purpose: Updates the status of an existing evaluation request.
  - Parameters:
    - *new\_status* (string): The updated status (e.g., "pending" → "running").
  - Returns: True if successful, False otherwise.
- ***get\_watermark\_information(registered\_images\_ids)***
  - Purpose: Retrieves watermark data for a list of registered image IDs.
  - Parameters:
    - *registered\_images\_ids* (list): IDs of images for which watermark info is requested.
  - Returns: Watermark metadata for the specified images.
- ***submit\_verification\_result(request\_id, verification\_data)***
  - Purpose: Submits verification results including watermark validation, similarity scores, and source identification to the blockchain and stores relevant data.



- Parameters:
  - `request_id`: The evaluation request ID.
  - `verification_data`: Includes watermark check result, similarity score, and other relevant metrics.
- Returns: Blockchain transaction ID or submission status.

The interfaces described above represent an initial concept for how the verification and registration system should function. While these definitions form the foundation of the PoC, their structure, function signatures, and implementation details may evolve during collaboration with Secublox. However, the overall goals and functional roles of the interfaces will remain consistent throughout development.

### 3.2 Watermarking System

After the creation of a new image, the first step in our pipeline is the watermarking of the image. We here describe our frequency-based image watermarking system.

As all watermarking approaches, we distinguish between two phases (Figure 2): watermark embedding (or insertion), which happens immediately after image creation, and watermark extraction (or verification), which happens when a content verification request is submitted.

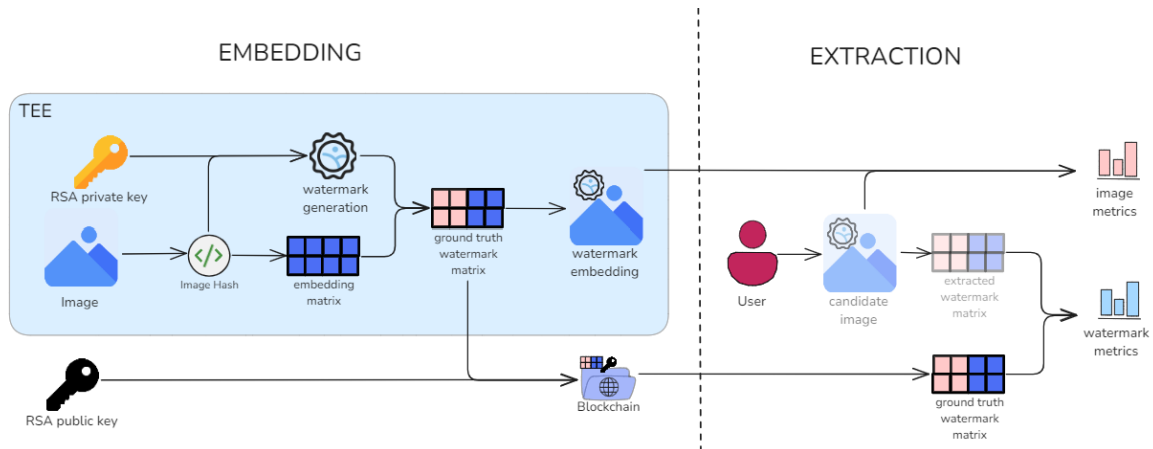


Figure 2: Image Watermarking System Diagram

Our system is based on the 2DWT-DCT approach proposed by Pathan and Yoshi [17]. The proposed approach works according to the following steps:

- *Watermark Generation with a Private Key*: a private key is used to generate the watermark, ensuring that the watermarking process is unique and secure. A hash of the image is signed with the private key, that signature is then converted into a unique watermark. Each watermark is unique for each user and image, making it also impossible for unauthorized users to recreate it.



- *Selection of Watermarking Frequency Positions:* the watermark is embedded in the latent space, on specific frequencies. Those frequencies are selected randomly using the hash of the original image, allowing secure further verification.
- *Watermark Embedding using the Watermark and the Watermark Positions:* for the watermark embedding (described in detail in the next sections), high frequencies are first filtered using a discrete wavelet transform (DWT), followed by zig-zag sampling and finally apply a frequency decomposition (DCT), where the watermark will be embedded and extracted. This process is similar to an encoder-decoder process, where the watermark is applied in the latent space. Using the encoder, we transform the visual-domain (i.e. pixels) representation of the image into a frequency-domain representation, also called *embedding*.
- *Reconstruction of the image:* after the embedding is done, the image is reverted into the visual domain, where the watermark is not perceptible.
- *Watermark Extraction with the Watermark Positions:* for later verification, it is possible to extract an existing watermark, by using the same encoding process, leading to the same latent space. Using the same positions, the watermark can be extracted.

This approach provides the following advantages:

- **High Security:** the private key ensures that the watermark is unique and cannot be easily replicated.
- **Improved Imperceptibility:** embedding in the LL sub-band of the wavelet transform ensures that visual quality is preserved.
- **Robustness Against Attacks:** the method withstands various attacks, including JPEG compression, noise addition, filtering, cropping, and sharpening.

The next sections describe the complete embedding (3.2.1 to 3.2.5) and extraction (3.2.6) steps in detail.

### 3.2.1 Preprocessing: 2DWT

The embedding process begins with a non-watermarked, three-channel (RGB) image. Discrete Wavelet Transform (DWT) transforms the image from the visual (“pixel”) domain into a multi-spectral representation, capturing frequency-like details with spatial locality. The output of a 2D DWT transform outputs sub-bands which contain directional detail information (LH, HL, HH) and a sub-band for approximation of coarse visual structure (LL). The first sub-bands are saved for future re-composition, but the one of interest for embedding is the latter one (LL), since it represents as well the low-pass filtered image.

First, DWT is applied on the original image, to generate the four sub-bands. A second round of DWT is applied on the low-pass filtered image (LL) to generate four additional sub-bands (called LH2, HL2, HH2, LL2).

The sub-band of interest for the later stages of the embedding is LL2, meaning that is the low-pass filtered image containing the lowest frequencies, which should stay consistent even under visual attacks.

This two-stage approach is the best trade-off between robustness and imperceptibility, by allowing the user to have a resilience toward noise attacks, without detecting the watermark with the human eye.

### 3.2.2 Preprocessing: Zigzag and sampling

During zigzag and sampling, we flatten the 2D matrix (LL2) from 2DWT into two 1D vectors, each with a similar information about the image.

A zig-zag exploration, as depicted in Figure 3, followed by odd and even sampling is performed to extract the diagonals of the image. By grouping items by even and odd index, the single 1D zigzag vector results in two different but highly correlated 1D vectors. The two vectors represent the odd and even diagonals, each having half of the pixels.

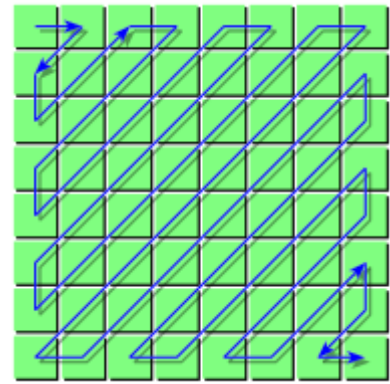


Figure 3: Exemplification of zig-zag pattern

### 3.2.3 Preprocessing: DCT

The best embedding space for the watermarking is not the visual domain, because it is too vulnerable to noise attacks. The frequency domain is a much more robust alternative to attacks.

Discrete Cosine Transform (DCT) is a performant frequency decomposition, but deliberate changes in the DC (0-frequency) coefficient can lead to large distortions in the visual property. Therefore, it is a better alternative to use DCT after 2DWT, as it retains the most important structural elements of the image, which are less likely to be impacted by an attack.

DCT-II, the most traditional decomposition, is applied on each of the two 1D vectors from the DWT space, to produce two 1D vectors that represent the even and odd zigzag vectors in the frequency space.

Since the two vectors before the frequency transform were highly correlated, i.e., the pixels they represented were very similar, they keep their high correlation also post-DCT transformation.

The DCT-II transformation completes encoding process, as the image is now translated into the embedding space where the watermark will be added.

### 3.2.4 Insertion of the watermark (“Embedding”)

Now that the preprocessing steps have been applied, and that two similar vectors are at disposal, the watermark is embedded on each of them.

The embedding process should allow future extraction, and possibly should not be reversible, meaning that the original image could be extracted by knowing the process. The former is a mandatory requirement, whereas the latter is optional.

The embedding is computed from a multi-channel (RGB) image, and since the embedding process was designed for a single-channel (grayscale) image, we apply the same technique on each channel separately. This shows strong results (see 4. Evaluation for more details.)

The first step requires to average the embedding frequencies of the vectors where the watermark is to be embedded. This averaging is necessary to have an extractable watermark in the future. By averaging the lower frequencies, there is a loss in the resulting image quality, however since the two vectors are almost similar, both have a very high correlation, so the loss is not very noticeable.

The next step is to add the watermark on the averaged frequencies of the vectors ( $x$  and  $y$ ). Both lower frequencies are similar, so we add the watermark on one vector and subtract the watermark on the other, resulting on slightly modified vectors ( $x' = x + 0.5 * w$ ,  $y' = y - 0.5 * w$ ). The idea behind doing this is that by computing the difference between the two vectors, the watermark can be found again, thus allowing future extraction ( $x' - y' = 0.5 * w + 0.5 * w = w$ ). This allows to extract the exact watermark by encoding the difference between the two vectors.

As mentioned before, the watermark can be extracted and, due to averaging, the original image cannot be recovered, making the process not fully reversible. However, someone who reverse-engineers the process, could extract the watermark easily and then use that watermark to remove/add from the vectors, thus retrieving vectors close to the original vectors pre-embedding. The vectors would slightly vary from the original ones because the low frequencies had been averaged before the embedding, but someone could reconstruct an image very similar to the original one, without any watermark. This however does not cause a problem to our ecosystem because the images are stored in our database the moment they are taken. Once in the database, we have further processes explained later, to protect the ownership of a picture.

### 3.2.5 Reversing the process to obtain the watermarked image

After embedding the watermark on the frequency space, the process is reversed, like with a decoder, to retrieve the embedded image in the visual space.

The following steps are applied in sequence:

1. Inverse DCT, i.e., the reverse process of DCT-II, also known as DCT-III, will revert the vectors from frequency space into the 2DWT space. The reconstruction is not lossless, but low enough not to impact the overall process.
2. Inverse zigzag and sampling, i.e., the inverse transformation of zigzag and sampling of Section 3.2.2. The 1D vectors contain the exact same shape and can thus be transformed back into a 2D matrix of the same shape as the original one, LL2. This inverse zigzag and sampling steps are lossless operations.
3. Inverse 2DWT, i.e., the LL2 2D matrix, now representing the second stage low pass filtered image, is now used with the previously generated sub-bands (LH2, HL2, HH2), to generate the first stage low pass image, LL. This LL matrix is used again with the previously generated sub-bands (LH, HL, HH) to reconstruct an image of the same shape as the original one.

At the end of the entire decoding process, an image similar to the original image is generated, with an imperceptible watermark embedded into it.

### 3.2.6 Watermark Extraction (during verification)

The extraction is the exact same encoding procedure stated previously, resulting in two similar 1D vectors that represent the frequency space.

Previously, the watermark was added on one and subtracted on the other vector, so to extract the watermark we simply subtract the two vectors, this should return a vector containing the watermark information. Since during the embedding, the watermark frequency positions were used alongside the watermark, here we also need the same watermark frequency positions to extract the watermark.

The watermark frequency positions are available publicly and can thus be easily retrieved for the verification process.

If there has been some alteration on the visual space due to some attacks, it might occur that the extracted watermark will not match exactly the original one used for embedding.

For verifying the robustness of the embedding technique against such alterations, we perform some attacks on the watermarked image, and then extract the watermark from it. We perform some metric analysis on the watermarks by comparing the original and extracted watermarks. The results are presented in Section 4 (Evaluation).

### 3.3 Trusted Execution Environment (TEE)

Our watermarking approach must be guaranteed to be protected from impersonation attacks, which would undermine the credibility of the watermark verification process. For this reason, the watermark embedding is designed to occur inside a Trusted Execution Environment (TEE).

As discussed in the Related Work section (2.3), a TEE is a secure area of a processor that ensures code and data loaded inside are protected with confidentiality and integrity. In this section, we discuss our use of TEE-related technologies. We first provide an overview of OP-TEE and its connection to related tools (QEMU, Rust, Teaclave TrustZone SDK), then in the last section (3.4.5) we discuss our approach for building a secure watermarking application running inside a TEE.

The foundation of a TEE lies in its separation at three critical hardware layers: the bus, the SoC core, and the debug interface. This design enforces three security guarantees: a) data confidentiality, i.e. ensuring that data processed within the TEE cannot be read by unauthorized parties; b) data integrity, i.e. preventing unauthorized data modifications; and c) code integrity, i.e. ensuring only trusted code executes within the TEE.

#### 3.3.1 OP-TEE

OP-TEE (Open Portable Trusted Execution Environment) [46] is an open-source implementation of a TEE, that complements the non-secure Linux kernel operating on ARM Cortex-A cores with ARM TrustZone technology. Its primary objective is to provide a secure environment, known as the secure world, for executing Trusted Applications (TAs). These applications are isolated from the main operating system, referred to as the Rich Execution Environment (REE) or *normal world*, ensuring strong security through hardware-enforced isolation.

OP-TEE achieves this isolation by ensuring that TAs operate independently of both the normal world and other TAs, leveraging the capabilities of ARM TrustZone or equivalent isolation mechanisms. It is designed with a small memory footprint, making it suitable for embedded and secure systems with constrained resources. The system is highly portable and can be adapted to various hardware architectures, supporting multiple client operating systems or coexisting TEEs [46].

#### 3.3.2 OPTEE with Rust

Rust, a programming language renowned for its safety guarantees and efficient performance, complements OP-TEE's secure architecture by enabling the development of robust Trusted Applications. By leveraging the Teaclave TrustZone SDK [47], developers can

create TAs in Rust that benefit from the language's memory safety, concurrency model, and modern tooling.

Rust offers several advantages for TEE development. Its compile-time checks eliminate vulnerabilities such as buffer overflows and null pointer dereferencing, ensuring memory safety. Additionally, Rust provides high performance with no runtime overhead for garbage collection, making it an ideal choice for resource-constrained environments. Modern tooling, such as Cargo for dependency management and Clippy for linting, further enhances the development experience.

In OP-TEE, Rust-based TAs can be developed in two primary modes [47]. The *no-std* mode is optimized for environments without the Rust standard library, prioritizing minimalism and performance. This mode produces small binary sizes and achieves faster execution but is limited in compatibility with third-party libraries that depend on the standard library. On the other hand, the *std* mode includes the standard library, enabling greater flexibility and access to a broader range of third-party libraries. However, maintaining compatibility with the latest Rust versions requires manual updates, which can be challenging. Rust-based TAs can be applied across various secure applications, including cryptographic operations and secure data storage. In combination with the robustness of Rust, OP-TEE serves as a powerful prototyping platform to build secure, efficient, and reliable applications tailored for secure embedded systems.

### 3.3.3 Emulation with QEMU

QEMU [48] is a versatile open-source emulator and virtualization software widely used in embedded systems development. It provides a virtualized environment to simulate hardware platforms, making it an invaluable tool for early-stage development and testing. For OP-TEE applications, QEMU allows developers to emulate the ARM architecture, specifically the *aarch64-unknown-linux-gnu* target, which matches the architecture of many supported devices, including the Raspberry Pi. By emulating this setup, we can build and test Trusted Applications (TAs) in a controlled and predictable environment without relying on physical hardware.

The primary advantage of QEMU is its ability to simulate hardware behavior with high fidelity. This enables the identification and resolution of potential issues in the early stages of development, saving time and resources. Furthermore, QEMU's integration with OP-TEE simplifies the process of running secure applications in a virtualized ARM TrustZone environment. Therefore, QEMU acts as a practical starting point to test and validate applications before deploying them to actual devices, ensuring that the software runs as intended on the target architecture.

### 3.3.4 Implementation on a Raspberry Pi

The Raspberry Pi [49] is widely used IoT device, which can be employed for developing and prototyping OP-TEE applications. It is a widely adopted, cost-effective device with robust community support and extensive documentation [49]. The Pi's reliance on the ARM architecture aligns perfectly with the requirements of OP-TEE, making it an ideal choice for testing secure applications in a real-world embedded environment.

One of the key benefits of the Raspberry Pi is its accessibility and flexibility [49]. One can easily transition from QEMU to the Raspberry Pi to test applications on physical hardware, allowing for validation of real-world performance and hardware interaction. Additionally, its rich ecosystem of peripherals and accessories supports diverse use cases, from IoT to edge computing, making it a versatile option for TEE-based applications.

By choosing the Raspberry Pi (models 3 and 3+), we could leverage its large community support for troubleshooting and optimization, speeding up development cycles. Moreover, its compatibility with OP-TEE ensures seamless execution of applications designed for the *aarch64-unknown-linux-gnu* target. The combination of QEMU for initial testing and the Raspberry Pi for hardware validation creates a streamlined development workflow, bridging the gap between virtualized environments and physical deployment.

### 3.3.5 Development and Testing Workflow for OP-TEE Watermarking

The workflow for developing and deploying our Rust-based Trusted Application (TA) for watermarking follows a structured approach, ensuring reliability and compatibility across virtual and physical environments. This process is designed to optimize the use of available resources and minimize potential errors before deploying on actual hardware.

#### 3.3.5.1 Developing and Building the Watermarking Code

The first step is to develop the watermarking application in Rust. This application was then compiled with the target architecture set to *aarch64-unknown-linux-gnu*, which corresponds to the ARM architecture used in OP-TEE environments. As the development environment on our local machines is based on the x86 architecture, cross-compilation was necessary to generate the appropriate binaries for the target platform.

During this compilation process, certain dependencies required manual intervention. Some libraries had to be downgraded to ensure compatibility, while others had to be manually compiled for the *aarch64-unknown-linux-gnu* target. These adjustments were crucial for resolving compatibility issues and ensuring the binary could run smoothly in the OP-TEE environment. Additionally, the development was carried out in `std` mode, as our application relies on several third-party crates that require the standard library. Using `std` mode allowed



us to take full advantage of these dependencies, which are critical for the watermarking functionality, without having to resort to the limitations of no-std mode.

#### 3.3.5.2 Testing on QEMU

After successfully compiling the application, the generated binary was first tested using OP-TEE on QEMU. By simulating the aarch64-unknown-linux-gnu architecture, QEMU provides a virtualized environment to emulate the Raspberry Pi's setup. This stage is invaluable for identifying and fixing potential issues early in the development cycle.

Testing on QEMU ensures that the application operates as intended within a trusted execution environment (TEE). Because QEMU closely replicates the behavior of the target architecture, it serves as an effective model for evaluating functionality, performance, and security. This initial testing phase significantly reduces the risk of encountering critical issues during hardware deployment.

#### 3.3.5.3 Deploying to the Raspberry Pi

Once the application passed all tests on QEMU, it was transferred to a Raspberry Pi for further validation. Both QEMU and the Raspberry Pi were set up with similar OP-TEE development environments, ensuring consistency between the virtual and physical platforms. This uniformity guarantees that the binary behaves in a similar manner on the Raspberry Pi as it does on QEMU.

The Raspberry Pi serves as the final testing ground, providing real-world hardware conditions to validate the application's performance and hardware interactions. The successful execution of the binary on the Raspberry Pi demonstrates the effectiveness of our pipeline, which first leverages the flexibility of QEMU for debugging and optimization before transitioning to actual hardware.

This iterative pipeline, from local development and cross-compilation to QEMU testing and Raspberry Pi deployment, provides a robust and efficient workflow for building secure applications using OP-TEE and Rust.

### 3.4 Multi-layered Semantic Integrity Verification

The watermarking system ensures that registered images can still be identified as the source of a given candidate image. However, it does not prevent semantic manipulations, i.e. attacks that alter the meaning of an image while preserving the overall pixel structure. Such manipulations include cropping, insertion, and contextual distortion, all of which may bypass traditional watermark verification while significantly changing the image interpretation.



Since an attacker has access to the watermarked image but cannot remove the watermark, they may attempt localized edits or content cropping. These attacks are particularly effective when the watermark remains intact, but localized edits distort the image intended message.

To address this, Deepshield integrates a multi-layered Semantic Integrity (SI) system, described in this section. The goal of the SI system is to ensure these changes are detectable and auditable whether they change the meaning or not. The SI system compares the original, watermarked image, with the candidate (or “retrieved”) image, which is processed during watermark verification.

### 3.4.1 Overall Architecture and Workflow

Our system for Semantic Integrity verification is multi-layered, i.e. it is composed of multiple layers of analysis which come to an aggregated conclusion by performing independent content evaluations. We call each of these layers a **semantic integrity tool**. For this phase of the project, we have devised four different tools: an image captioning tool based on text embedding similarity (3.4.2), an object detection tool based on bounding box comparison (3.4.3), an visual embedding tool based on comparison in the embedding representation space (3.4.4), and a key-point detection tool (3.4.5) based on Oriented FAST and Rotated BRIEF (ORB) key-point matching [50]. Figure 4 depicts a diagram of our semantic integrity verification system.

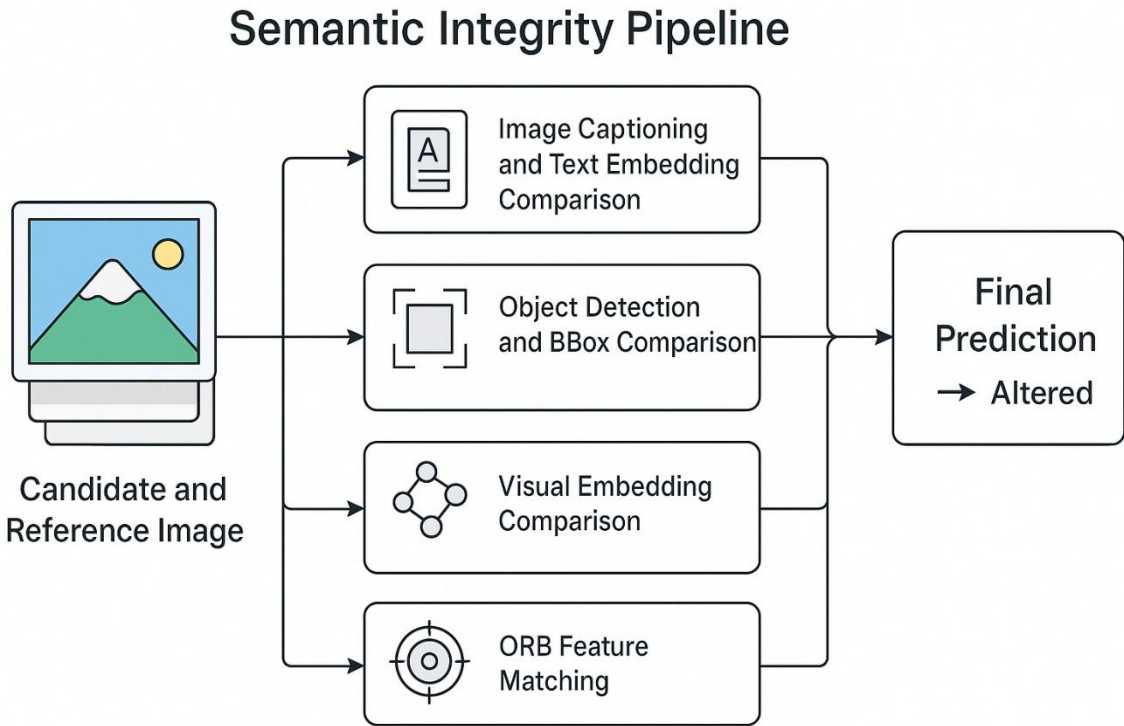


Figure 4: Architecture Diagram of the Semantic Integrity Evaluation System

### 3.4.2 Tool 1: Image Captioning and Embedding Similarity

The image captioning tool relies on advanced vision-language models (VLMs) to understand and describe the context of the candidate and reference images. We use two main model components: 1) **a captioning model** (a VLM, like *Qwen2.5-VL-7B-Instruct* [51]), which produces a textual description of an image, and 2) **a text embedding model** (such as *Jina Embeddings v2 base-en* [52]), which transforms a piece of text into a latent vector representation, encoding meaning and context.

First, we generate text captions for both the candidate and reference images. These captions reflect each image's visible content, such as the setting, the characters and object present in the scene and their relationship, and most importantly, the implied meaning or message of the two images.

We then convert the two captions into vector embeddings using the text embedding model, to measure how closely they match. This step encodes the semantic information from the captions into a format that can be quantitatively compared. If the semantic meaning has significantly changed, such as removing a key object or altering the setting, the caption embeddings will diverge.

A similarity metric (commonly cosine similarity, ranging from 0 to 1) is computed between the two caption embeddings. If the similarity score drops significantly ( $< 1$ ), it indicates a strong conceptual mismatch. It shows that key elements in the image have been altered or removed.

By using a caption-based approach, the system captures more than raw pixel data. It interprets and evaluates the scene at a conceptual level. When the captions differ substantively, it often indicates a genuine change in meaning rather than just superficial edits. However, the reliability of this method depends heavily on the quality of the captions produced by the VLM. If the model fails to mention critical details, small but significant manipulations may not be flagged. In this sense, the choice of the system prompt used to generate captions is fundamental.

### 3.4.3 Tool 2: Object Detection

As a second tool, we implemented a technique that relies on object detection and bounding-box analysis to check whether the main elements in an image have been preserved. This method identifies each object present in the reference image, locates the same objects (if they exist) in the candidate image, and then quantifies how consistently they appear across both versions.

First, an object detection model, such as *Qwen2.5 VL 7B* [52], is used to identify and label objects in both the reference and candidate images. The output includes bounding-box

coordinates (e.g., [x min, y min, x max, y max] [x min, y min, x max, y max]) and class labels (e.g., human, animal, object, background) for each detected object.

Matched objects (those sharing the same label in both images) are evaluated by calculating the Intersection-over-Union (IoU) between their bounding boxes. This measure indicates how much of the object remains spatially consistent. A high IoU means the objects occupy similar regions in both images, while a low IoU suggests that they have been removed, severely cropped, or relocated.

A global score for the similarity of the two images is then derived by averaging IoUs for all matched objects, providing an overall sense of how faithfully the modified image retains the semantic elements of the original scene.

This approach is particularly effective for detecting manipulations like cropping out parts of the image, hiding or erasing objects, or shifting their positions to alter the narrative. While it is more granular than a strictly embedding-based approach, it does rely on accurate object detection. If the detector fails to recognize certain objects or mislabels them, the comparison may overlook some manipulations.

#### **3.4.4 Tool 3: Image Embedding Comparison**

Image embeddings are latent vector representation similar to text embeddings, which however represent the meaning of images in a quantitative form. Our third method involves generating image embeddings that capture the overall semantic content of each image. In this approach, images are not described verbally or through bounding boxes; instead, they are converted into feature vectors that reflect high-level scene understanding and object relationships.

First, models like *DINOv2* [53] and other variants are used to encode each image into a dense feature vector. These embeddings aim to capture a wide spectrum of information, including object identity, spatial relations, and to some extent even stylistic elements of the scene.

Once the embeddings for the original and the altered image have been obtained, their cosine similarity is computed. As for image captioning, high similarity score (close to 1) indicates that the overall semantic representation remains nearly the same, whereas a significantly lower score suggests that a notable semantic change has occurred.

Since this method treats the two images as a whole, minor structural edits or small cropping might not drastically affect the similarity if the overall scene remains consistent. However, large or context-altering modifications will generally cause a visible drop in embedding similarity.

Embedding comparisons of images can be performed quickly and often require fewer model components than captioning or bounding-box-based methods. The trade-off is that one

gains less interpretability, knowing that the embedding changed does not immediately explain which part of the image was altered or why.

### 3.4.5 Tool 4: ORB Feature Matching

As a fourth tool, we implement a traditional computer vision approach based on keypoint detection and feature matching using the ORB algorithm [50]. Unlike embedding-based methods or object-level reasoning, this technique compares two images by analyzing the spatial consistency of low-level visual features. The method is especially effective for identifying structural alterations such as cropping, shifting, or object removal, and works well when the manipulations do not fundamentally alter the semantic content but still affect geometric or textural integrity.

The ORB (Oriented FAST and Rotated BRIEF) algorithm is a well-established method for detecting and describing local keypoints in images. These keypoints represent visually distinctive areas, such as corners or edges, and are invariant to scale and rotation, making them suitable for robust image comparison. In our pipeline, ORB is applied to both the reference image and the candidate image to extract sets of keypoints and associated descriptors.

Once keypoints are extracted, the system performs feature matching using a Brute Force matcher with Hamming distance, followed by Lowe’s ratio test to filter ambiguous matches. This ensures that only high-confidence matches are considered when assessing similarity. The total number of these “good matches” provides a quantitative basis for evaluating image integrity. If the number of matches falls below a configurable threshold, it is a strong indicator that meaningful parts of the image may have been removed, moved, or obscured.

A key strength of this method is its simplicity and interpretability: it provides direct information about how many visual features remain consistent between the two images. Moreover, it does not rely on deep learning models or captions, making it suitable for resource-constrained environments or deployment on devices where model inference is expensive or unavailable.

However, the ORB-based approach has limitations. It may be less effective in detecting semantic-level changes that preserve visual structures (e.g., replacing one person with another in the same pose), and it is more sensitive to blur, illumination changes, or artistic filters. Additionally, the tool assumes that enough high-quality keypoints are present in both images; otherwise, feature matching becomes unreliable.

Overall, this method serves as a lightweight, geometry-aware baseline that complements more abstract methods such as embeddings or captioning. It enables fine-grained analysis

of spatial consistency and is particularly valuable in scenarios where image authenticity must be verified without reliance on high-level semantics.

#### **3.4.6 Response aggregation and Final Comments**

After individual tools have evaluated the semantic integrity of an image pair, their outputs are combined using an aggregation policy. This step synthesizes the multiple perspectives of the tools (textual, visual, spatial) into a unified decision. Available policies include majority voting, mean thresholding, minimum or maximum logic, each catering to different trade-offs in sensitivity and specificity. For example, our default choice majority voting offers a balanced outcome, while a minimum-based policy enforces stricter consistency across all tools.

This modular aggregation layer enhances robustness by allowing the system to mitigate individual tool weaknesses through consensus. It also provides transparency into each tool's contribution, supporting traceable and explainable integrity assessments.

Our semantic integrity checking framework is designed with extensibility and modularity in mind. New tools can be integrated with minimal effort. This allows us to easily swap models, test custom comparators, or introduce novel approaches without altering the overall evaluation pipeline. Potential expansions include incorporating EXIF metadata analysis to detect inconsistencies in image provenance, or advanced scene graph reasoning for deeper understanding of inter-object relationships and implied narratives.

## 4 Evaluation

In this chapter, we evaluate our approach, then present and discuss the results. First, we introduce the methodology of our evaluation, including employed hardware, datasets, metrics. Then we illustrate the evaluation results. Finally, we draw conclusions and takeaways based on the analysis of the results.

### 4.1 Experimental Setup

This section describes the technical environment used for developing, deploying, and testing our full image verification system. This section focuses on the involved hardware and software, which is shared by all components of our system, namely watermarking and semantic integrity. The specific details about the evaluation methodology of each component, such as metrics and datasets, are presented close to the corresponding evaluation results (4.2.1 and 4.3.1).

In general, our evaluation environment spans embedded and high-performance computing platforms to simulate realistic deployment scenarios, ranging from edge devices to centralized AI inference.

- **Hardware:** The setup includes two Raspberry Pi devices (a Pi 3 and a Pi 3+), with the Pi 3 running OP-TEE and the Pi 3+ running Raspbian for client-side operations. A GPU server equipped with an NVIDIA L40S 46GB GPU is used for running the AI-based semantic integrity tools. Additional development and testing were performed on local laptops.
- **Software:** The TEE-side watermarking and image-handling logic is implemented in Rust and C, running inside QEMU during development and on-device in deployment. The surrounding infrastructure, including capture, encryption, and communication logic, is implemented in Python, with dependencies managed via Poetry to support a clean development cycle.
- **Model Serving:** All machine learning models for semantic integrity checking are served via an HTTP API hosted on the GPU server, decoupling the image analysis logic from resource-constrained edge devices.

We implemented the full system architecture as described in Section 3. For test purposes, the Raspberry Pi 3+ (non-OPTEE) captures a picture, encrypts it, and sends it to the OP-TEE-enabled Raspberry Pi 3 for watermarking. The watermarked image is then returned to the Pi 3+ for registration and semantic integrity verification. The integrity analysis is performed by querying the API on the GPU server, which hosts the vision-language and embedding models, due to their computational requirements.

## 4.2 Watermarking Evaluation Results

### 4.2.1 Watermarking Evaluation Methodology

Here we discuss our evaluation framework for the watermarking system.

In image watermarking, imperceptibility and robustness are key characteristics. Imperceptibility ensures that embedding a watermark does not degrade the visual quality of the host image, while robustness guarantees that the watermark can be accurately recovered even when the watermarked image undergoes various attacks or distortions.

We have evaluated our watermarking approach under a range of distortion types at different levels of watermark strength coefficient (denoted by  $\alpha$ ), which determines how aggressively the watermark signal is visually embedded into the image. In general, higher  $\alpha$  can improve robustness but at the cost of increased visible artifacts.

We tested the watermarking system on the MS-COCO 2017 validation dataset, containing 5000 images. We examined how each distortion, such as blurring, compression, noise addition, contrast adjustment, brightness adjustment, rotation, and resized cropping, impacts both the image quality and the watermark recovery accuracy.

We rely on multiple metrics to capture the diverse aspects of watermarking performance:

1. **Peak Signal-to-Noise Ratio (PSNR)**: measures the difference between the original (unwatermarked) image and the watermarked or distorted image. A higher PSNR indicates minimal distortion (good imperceptibility). A drop in PSNR typically means more visible artifacts in the image. A PSNR above 30 dB is usually considered acceptable.
2. **Structural Similarity Index (SSIM)**: evaluates the perceptual similarity between two images by considering luminance, contrast, and structure. SSIM values range from -1 to 1, where 1 represents an identical image. Values below 0.8 often indicate noticeable quality degradation.
3. **Bit Correction Rate (BCR)**: represents the percentage of correctly recovered watermark bits after distortion. A high BCR (close to 100%) signifies strong robustness against attacks. A significant drop in BCR indicates that the watermark cannot be reliably extracted.
4. **Normalized Cross-Correlation (NCC)**: measures how closely the extracted watermark matches the original watermark in terms of spatial correlation. NCC values near 1 indicate near-perfect alignment; below 0.8 usually indicates poor recovery quality.
5. **Correlated Aligned Transform Sign Overlap (CATSO) Score**: a custom metric devised to quantify the sign consistency between the original watermark's bits and

those extracted after distortion. It ranges from -100 to 100. Scores near 100 signify near-perfect bit alignment; negative scores mean significant sign flipping (poorer recovery).

Below, we summarize the key observations for each of our monitored watermark metrics, when varying the watermark strength coefficient  $\alpha$  across different distortions.

#### 4.2.2 PSNR vs. $\alpha$

Figure 5 shows the effect of changing watermark strength coefficient  $\alpha$  on the Peak Signal-to-Noise Ratio (PSNR) metric. In general, we observe PSNR decreases with higher  $\alpha$ . This is expected because a stronger watermark injection typically introduces more visible distortions.

Blurring and compression distortions show a more pronounced drop in PSNR, indicating these distortions amplify the visible artifacts introduced by the watermark. Noise and brightness adjustments retain relatively higher PSNR across varying  $\alpha$ , suggesting minimal impact on perceptual quality in these cases.

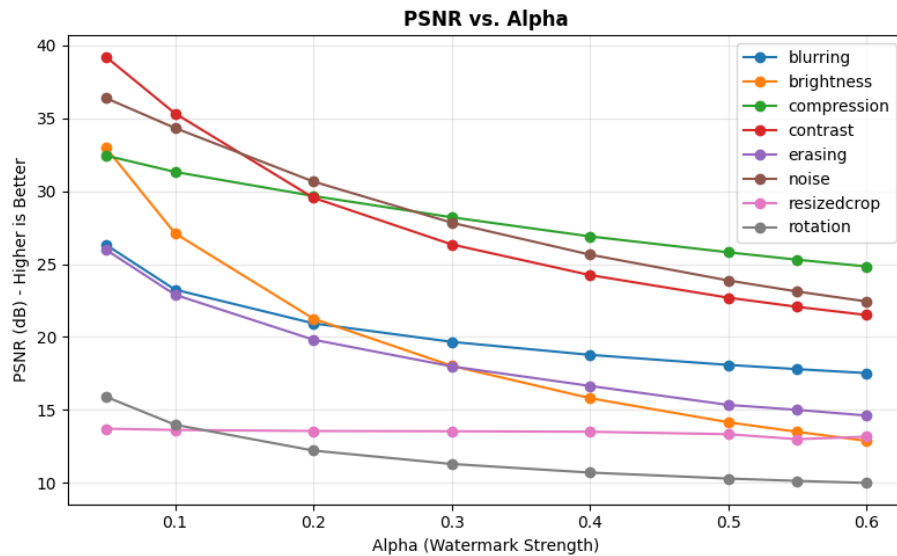


Figure 5: effect of watermark strength coefficient  $\alpha$  on the Peak Signal-to-Noise Ratio (PSNR) metric.

#### 4.2.3 SSIM vs. $\alpha$

Figure 6 depicts the impact of  $\alpha$  on the Structural Similarity Index (SSIM) metric.

In general, we observe how SSIM decreases as  $\alpha$  increases, reflecting growing perceptual differences from the original image. Compression and rotation distortions exhibit the steepest declines in SSIM, suggesting these attacks significantly affect perceptual similarity.



Noise and contrast adjustments maintain higher SSIM values, implying that even at higher  $\alpha$ , the image retains a closer perceptual appearance to the original.

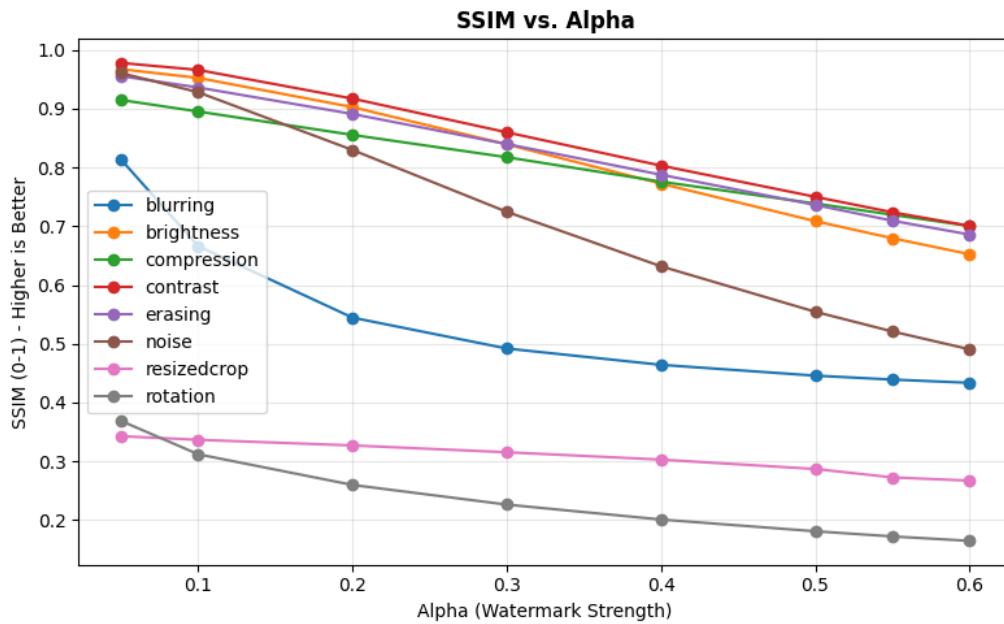


Figure 6: effect of watermark strength coefficient  $\alpha$  on the Structural Similarity Index (SSIM) metric.

#### 4.2.4 BCR vs. $\alpha$

Figure 7 depicts the impact of  $\alpha$  on the Bit Correction Rate (BCR) metrics. In general, the watermark robustness (as measured by BCR) often improves with a slight increase in  $\alpha$  up to a certain point, because the watermark is more deeply embedded. However, certain distortions can negate these gains if they destroy the embedded signal.

In terms of specific distortions, noise alterations still produce a near-perfect 100% BCR for all  $\alpha$  values, indicating a high resilience of the watermark against typical noise addition.

Blurring and rotation distortions lead to significant drops in BCR at higher  $\alpha$  values. The spatial transformations from blurring and rotation disrupt the watermark structure more severely. Compression and contrast adjustments, on the other hand, show minimal loss in BCR, staying close to 100%.

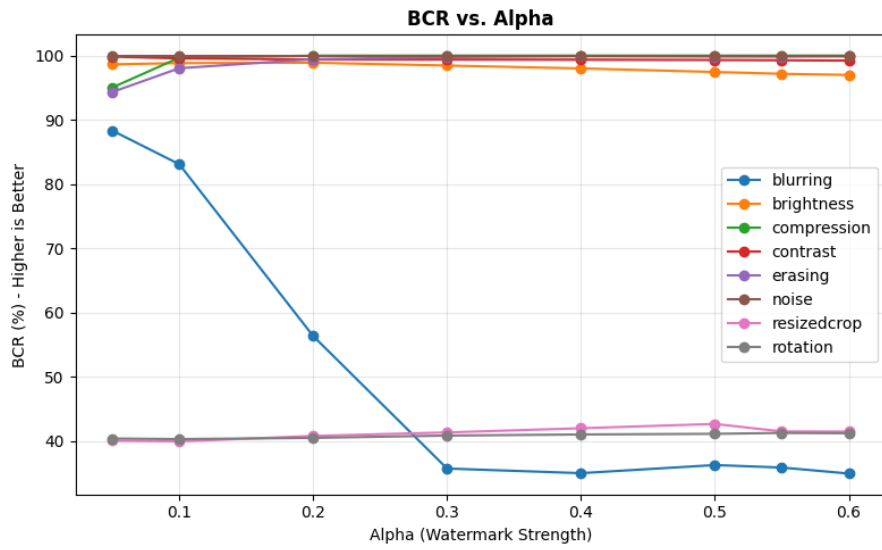


Figure 7: effect of watermark strength coefficient  $\alpha$  on the Bit Correction Rate (BCR) metric.

#### 4.2.5 NCC vs. $\alpha$

Figure 8 depicts the effect of watermark strength coefficient  $\alpha$  on the Normalized Cross-correlation (NCC) metric. NCC tends to remain high at lower  $\alpha$  but can decrease when  $\alpha$  becomes very large, especially if the distortion heavily alters the embedded signals.

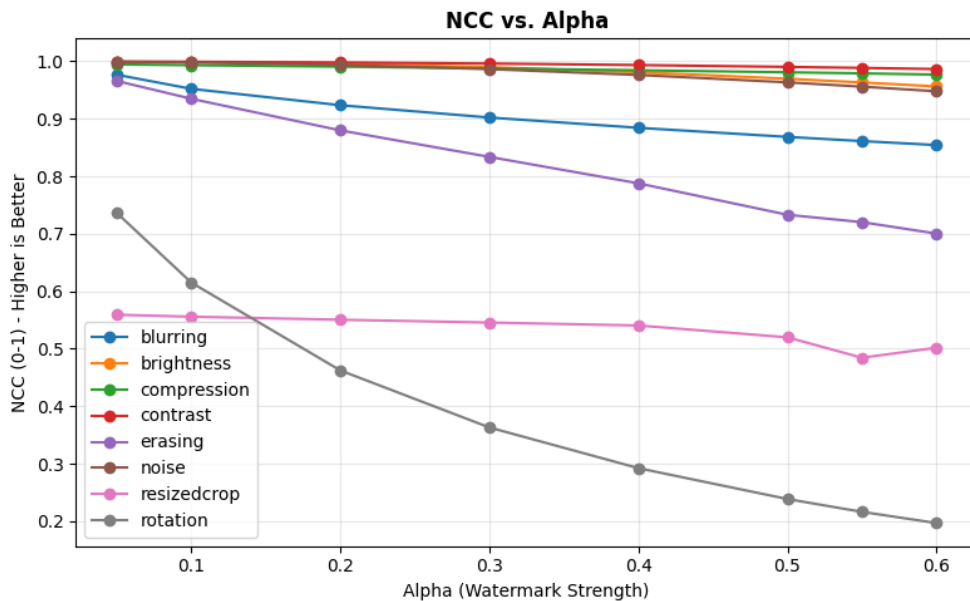


Figure 8: effect of watermark strength coefficient  $\alpha$  on the Normalized Cross-correlation (NCC) metric.

Blurring and rotation alterations cause notable reductions in NCC at higher  $\alpha$ . Compression, noise, and contrast largely maintain high NCC, indicating strong robustness to these types of attacks.

#### 4.2.6 CATSO Score vs. $\alpha$

Figure 9 depicts the effect of watermark strength coefficient  $\alpha$  on the Correlated Aligned Transform Sign Overlap (CATSO) metric.

In general, higher  $\alpha$  often helps maintain bit alignment, provided the distortion does not disrupt the watermark’s positional consistency.

Noise distortions continue to show an ideal CATSO score (~100), illustrating excellent resilience to random pixel fluctuations. Rotation and resized cropping alterations frequently lead to negative or near-zero CATSO scores at higher  $\alpha$ , suggesting many watermark bits become flipped or misaligned. Compression and contrast alterations show stable CATSO scores, implying minimal sign mismatch during watermark extraction.

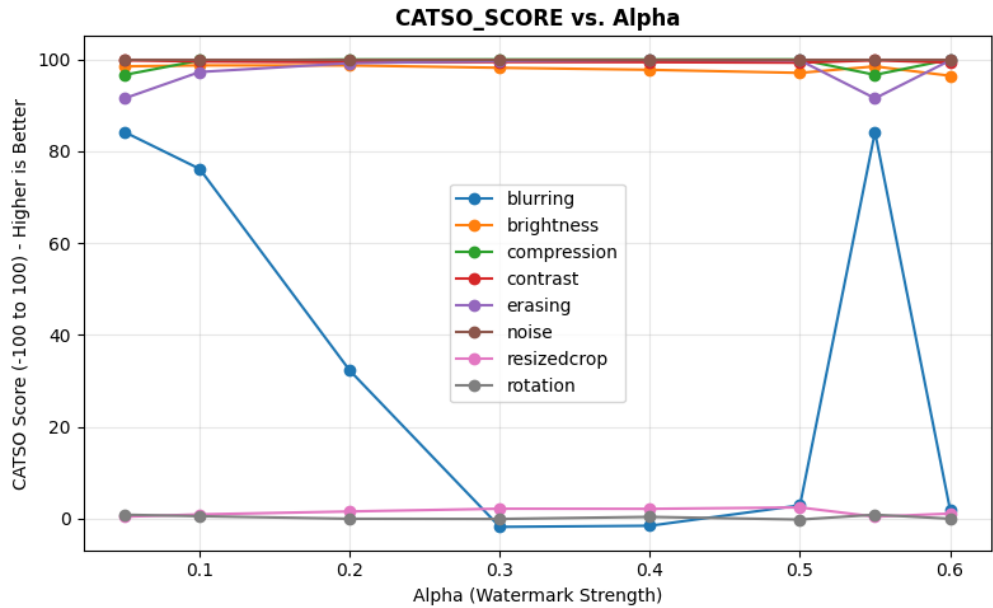


Figure 9: effect of watermark strength coefficient  $\alpha$  on the Correlated Aligned Transform Sign Overlap (CATSO) metric.

#### 4.2.7 Summary of Watermarking Results

In our comprehensive watermarking evaluation, we analyzed the impact of various distortions and watermark strength levels (denoted by  $\alpha$ ) on both visual quality and watermark recovery performance using five key metrics: PSNR, SSIM, BCR, NCC, and CATSO. The main findings are summarized in this section.

The proposed watermarking approach demonstrates a **strong balance between visual quality and robustness under a wide range of distortions**. Our evaluation, based on five key metrics confirms the effectiveness of the method, particularly under common image manipulations. Below, we summarize the most relevant observations, highlighting both the strengths of the approach and opportunities for improvement.

A moderate watermark strength ( $\alpha \approx 0.04\text{--}0.06$ ) **consistently yields the best compromise between invisibility and resilience**. Within this range, the watermark remains largely imperceptible, with PSNR values above 32 dB and SSIM scores exceeding 0.9 across most benign distortions. This ensures minimal visual degradation while still embedding a watermark signal strong enough for reliable extraction.

The watermarking method shows **excellent robustness against additive noise and compression artifacts**. Even at low  $\alpha$  values, recovery metrics remain close to perfect: BCR and NCC both approach 100%, and CATSO scores consistently remain above 95%. This confirms that the embedded watermark can survive practical conditions such as image saving and transmission without significant degradation.

**Contrast and brightness alterations have negligible effects** on both perceptual quality and recovery performance. PSNR and SSIM metrics remain high ( $>33$  dB,  $>0.92$  respectively), and watermark integrity is preserved (BCR  $> 98\%$ , NCC  $> 0.98$ , CATSO  $> 95\%$ ). These findings support the suitability of the method for real-world usage scenarios involving moderate editing.

**Blurring, rotation, and resized cropping introduce more severe challenges**. In these cases, BCR drops below 80% at high  $\alpha$ , and CATSO scores can fall below 50%, indicating bit misalignment and partial loss. While increasing  $\alpha$  improves resilience in some cases, it comes at a perceptual cost, with PSNR falling below 30 dB and SSIM dropping below 0.8. These results highlight the need for enhanced embedding strategies or synchronization mechanisms to better withstand geometric transformations.

For applications where high perceptual quality is essential, and only light distortions are expected (e.g. social media sharing, web publication), a conservative  $\alpha$  setting ( $\approx 0.04$ ) is recommended. In environments where robustness to more aggressive distortions like compression or moderate blurring is necessary, slightly higher  $\alpha$  values ( $\approx 0.06\text{--}0.08$ ) may be adopted, provided the drop in image quality remains acceptable. We still retain the use of higher alphas (e.g.,  $\alpha \approx 0.2$ ) for visual testing purposes in our pipeline.

### 4.3 Semantic Integrity Evaluation Results

In this section, we present the results of the evaluation of our Semantic Integrity (SI) evaluation system. We present the methodology (4.3.1), the full results (4.3.2), and a summary of the conclusions from the results (4.3.3).

### 4.3.1 Semantic Integrity Evaluation Methodology

We here discuss the evaluation methodology for the semantic integrity verification system.

We employ the CASIA Image Tampering Detection Evaluation Database v2.0 [54] as our benchmark dataset. This dataset comprises 7491 authentic and 5123 tampered images, categorized into two primary tampering types: 1) **splicing**, where content from one image is inserted into another, and 2) **copy-move**, where a region within the same image is duplicated and pasted elsewhere. For our evaluation, we utilize a subset of 4978 samples where the ground truth was easily accessible, ensuring a balanced representation of both manipulation types.

Our evaluation treats the semantic integrity task as a binary classification problem, where each image is labeled as either altered (1) or non-altered (0). To assess the performance of our system, we compute standard metrics for binary classification, including **accuracy**, **precision**, **recall**, **F1-score**, and **true-negative rate (TNR)**. These metrics provide insights into the system ability to correctly identify tampered images and minimize false detections. We compute these metrics for each tool and for the combined prediction (majority-voting).

To accurately measure precision, we adopt a negative sampling strategy. This involves passing the same original image twice through the system, serving as a control group with a ground truth label of 0. This approach helps in evaluating the system's tendency to produce false positives. Furthermore, we analyze the confusion matrix components: true positives (TP), false positives (FP), false negatives (FN), and true negatives (TN), for each manipulation type. This detailed breakdown allows us to understand the system's performance nuances across different forgery categories and to identify areas for potential improvement.

### 4.3.2 Semantic Integrity Results

Table 2 summarizes the performance of our semantic integrity verification system across different tools and tampering types.

In the splicing scenario, all tools demonstrated high effectiveness. The image embedding tool achieved perfect scores across all metrics, indicating its robustness in detecting spliced content. The object detection tool and ORB feature matching tool also performed well (F1-scores of 0.9385 and 0.9503, respectively). The image captioning tool, while slightly less accurate, still maintained a strong F1-score of 0.8922. The combined approach, utilizing majority voting across all tools, yielded an F1-score of 0.9733, showcasing the advantage of integrating multiple detection methods to enhance overall performance.

Detecting copy-move forgeries proved more challenging. The image captioning tool led individual tools with an F1-score of 0.6779, while the object detection tool followed with 0.5375. The image embedding tool, and the ORB feature matching tool struggled in this

context, with F1-scores of 0.336 and 0.1103, respectively. The combined approach improved upon individual performances, achieving an F1-score of 0.5199. This suggests that while individual tools may falter, their integration can provide a more balanced detection capability.

Tool (or tool set)	Tampering	Accuracy	Precision	Recall	TNR	F1	Total	Pos.	Neg.
Combined	splicing	0.9726	0.9486	0.9994	0.9458	0.9733	3504	1752	1752
Image Captioning	splicing	0.8799	0.8091	0.9943	0.7654	0.8922	3504	1752	1752
Object Detection	splicing	0.9375	0.9233	0.9543	0.9207	0.9385	3504	1752	1752
Image Embedding	splicing	<b>1.0000</b>	<b>1.0000</b>	<b>1.0000</b>	<b>1.0000</b>	<b>1.0000</b>	3504	1752	1752
ORB Feature Matching	splicing	0.9484	0.9093	0.9960	0.9007	0.9507	3504	1752	1752
Combined	copy-move	0.6634	0.9060	0.3645	0.9622	0.5199	6452	3226	3226
Image Captioning	copy-move	<b>0.6927</b>	0.7121	<b>0.6469</b>	0.7384	<b>0.6779</b>	6452	3226	3226
Object Detection	copy-move	0.6427	0.7621	0.4151	0.8704	0.5375	6452	3226	3226
Image Embedding	copy-move	0.6001	<b>1.0000</b>	0.2002	<b>1.0000</b>	0.3336	6452	3226	3226
ORB Feature Matching	copy-move	0.5250	0.8675	0.0589	0.9910	0.1103	6452	3226	3226
<b>Combined</b>	<b>All</b>	<b>0.7722</b>	<b>0.9310</b>	<b>0.5880</b>	<b>0.9564</b>	<b>0.7208</b>	<b>9956</b>	<b>4978</b>	<b>4978</b>

Table 2: performance results of semantic integrity on CASIA V2.0 [54], grouped by tool and tampering type

Overall, the combined system achieved an accuracy of 0.7722 and an F1-score of 0.7208 across both manipulation types. The high precision (0.9310) indicates a low false positive rate, while the recall (0.5880) highlights areas for improvement in detecting all instances of tampering. The TNR (0.9564) further confirms the system's strength in correctly identifying unaltered images.

### 4.3.3 Summary and Conclusions

The evaluation underscores the effectiveness of our semantic integrity verification system, particularly in detecting splicing forgeries. The integration of multiple tools through majority voting enhances detection capabilities, compensating for individual tool limitations. However, the system's performance in identifying copy-move forgeries indicates a need for further refinement, possibly through the incorporation of additional detection methods or the enhancement of existing ones.

In conclusion, our modular framework demonstrates strong potential in semantic integrity verification, especially when leveraging the strengths of diverse detection tools. Future work will focus on improving detection accuracy for more subtle manipulations and expanding the system's capabilities to handle a broader range of tampering techniques.

## 4.4 Overall Summary and Takeaways

Our evaluation demonstrates that the proposed system achieves a practical balance between robustness and flexibility, across both watermarking and semantic integrity tasks. On one hand, the watermarking component proves effective under a wide range of visual distortions, with metrics like PSNR, SSIM, and BCR confirming the quality and recoverability of the embedded signal. Moderate watermark strength values provide an optimal compromise between imperceptibility and resilience, making the solution suitable for both high-fidelity and robustness-critical scenarios.

On the other hand, the semantic integrity verification system achieves high precision and strong results for splicing detection, especially when leveraging a combination of multiple detection tools through majority voting. The performance on copy-move manipulations, while more limited, highlights the difficulty of this manipulation type and points to opportunities for improvement. Taken together, the evaluation validates the soundness of our architecture and motivates future enhancements, such as extending the system with scene reasoning and improved tool vote aggregation.

## 5 Challenges

This section summarizes our challenges and open problems, while implementing and testing our Deepshield system. This section serves both as a retrospective on our execution as well as a record for future improvement and next steps to carry out in future potential phases of the project.

### 5.1 Watermarking

#### 5.1.1 Python implementation

The research paper that was the basis for this code implementation had a MATLAB implementation at disposal. However, we encountered several problems with this implementation, including the inadequate quality of the code, the confusing naming conventions, the facts that the implementation worked only for squared grayscale images.

We reimplemented the same functionalities in Python; therefore, our first challenge was to convert this MATLAB code into a generally functioning code with tests (including support for RGB arbitrary-size images). This proved to be a success as the evaluation metrics showed promising results for both image and watermark similarity.

#### 5.1.2 Rust implementation

The watermark embedding had to be done on the TEE device, meaning that it required a C or Rust implementation. One solution would have been to convert the Python implementation into a binary. This solution is not ideal, because we would have needed to call this underlying library inside C or Rust.

We decided instead to implement the same technique in Rust so that we would also have the compatible code running in the server to extract the watermark.

This turned out to be more time consuming than initially expected due to the inherent complexity of Rust, but the implementation was ultimately completed successfully. There were several challenges faced along the way.

The first big challenge was to have an exactly similar implementation in both Rust and Python, meaning that the underlying libraries had to have a similar behavior.

The zigzag, sampling and DCT were accurately performed in Rust. However, DWT turned out to be more challenging. Several libraries were tried out to mimic the Python behavior of DWT without success. Further exploration of the Python *pywt* library revealed that its core implementation was done in C. The subsequent approach was to leverage this C code by



creating Rust bindings. Although this integration posed challenges, it eventually yielded results matching those of the original implementation.

The embedding and extraction were then executed correctly on both Python and Rust, but needed more thorough examination, like the testing pipeline created in Python and with the same dataset. There was also a risk of having a slightly variation in Python and Rust code, which would create a slight variation in embedding with the Rust implementation and extraction with the Python implementation. Due to both reasons, Python bindings around the Rust implementation were created, so that the code could be evaluated using the same Python pipeline and same dataset. The embedding could then also be performed on a server using the exact same method implemented in Rust, thereby eliminating any potential for future discrepancies between embedding and extraction.

The final project is now much more complex than expected at first, resulting in a Python server calling Rust code calling C code. This mix of technology fulfils the requirements for a TEE device and mitigates potential problems, while keeping a good overall performance.

### **5.1.3 Evaluation**

To have a proper evaluation method, MS-COCO dataset has been used for benchmarking the different techniques. We implemented an evaluation pipeline that would compare each embedding implementation. It turned out quite early in the project that 2DWT-DCT is the most efficient technique, by leveraging low-pass filtering and then frequency embedding, which yields the best results in term of attack robustness and imperceptibility.

### **5.1.4 Key signature and verification**

In our initial approach to watermark generation, a private key was used for generation, with the intention to use the public key during the extraction to validate the ownership.

The watermark generation is taking a chunk of the signature to create a watermark of the desired length, the chunk is the bit size of the watermark. This however was a problem because the bit size of a key is way too big compared to the bit size of the watermark, hence there cannot be a recovery of the signature at extraction. If we were to use a compression method to match the watermark size, during extraction we could decompress the extracted watermark and obtain the signature that could be validated with the public key.

The SHA256 schema was initially used, meaning that the 256 bytes contains 2048 bits, whereas the watermark length is 255 bits. There is no compression schema that can reduce 10 folds with perfect recovery. There needs to be more exploration in the watermark generation technique to obtain a fully reversible process which would allow verifying the watermark with the public key.

## 5.2 Trusted Execution Environment

### 5.2.1 QEMU

One of the most significant hurdles was the need to cross-compile several libraries used in the watermarking code for the architectures supported by OP-TEE. These architectures, such as aarch64-unknown-linux-gnu, differ from the x86 architecture of our local development machines. Cross-compilation introduces intricacies because many libraries are not inherently designed to support these target architectures.

For some dependencies, precompiled binaries are unavailable or incompatible, necessitating the manual compilation of these libraries. This process involves configuring build environments, ensuring the correct toolchains are installed, and adapting build scripts to produce binaries suitable for the target architecture. Any misstep in this process could result in incompatibilities or runtime errors in the Trusted Application.

The OP-TEE environment supports a specific version of Rust and its standard library, which often lags behind the latest stable releases of Rust. For instance, the version supported might be several months or even years older than the version used in local development environments. This mismatch creates compatibility issues, especially when the watermarking application or its dependencies rely on features or APIs introduced in newer Rust versions.

These version constraints mean that developers must carefully align their codebase with the supported Rust version. This alignment often involves identifying dependencies that are incompatible, removing or replacing them, and refactoring code to avoid unsupported features.

As the watermarking code evolves, new dependencies are introduced to enhance functionality or optimize performance. Each new dependency must be scrutinized to ensure compatibility with the OP-TEE-supported architecture and Rust version. This leads to additional work, including downgrading dependencies, modifying code to use alternative crates, or manually patching libraries to function within the OP-TEE environment.

### 5.2.2 OP-TEE and Raspberry Pi

Although QEMU was used to support the development and verification of the watermarking code written in Rust, the final execution of the watermarking code must take place on an OP-TEE-enabled Raspberry Pi, which needs to be correctly configured. This introduced a series of additional challenges and efforts.

One major challenge was the additional setup required to build and deploy OP-TEE on the Raspberry Pi. Unlike traditional development environments, OP-TEE for Raspberry Pi mandates low-level tooling and potentially risky commands like *make img-help*, which is used

to erase and flash partitions on the SD card. This step can be dangerous if not properly isolated, as it risks affecting local desktop partitions during experimentation. Moreover, despite documentation indicating support for common communication methods to the OPTEE Raspberry Pi such as UART, WiFi, and NFS, these were not functional out of the box. As a workaround, SSH was manually installed and started on the Pi, but it required explicit network configuration with command-line on every boot, due to the absence of systemd or persistent network services.

Another obstacle involved ensuring compatibility between the OP-TEE environment on QEMU and the one running on the physical Raspberry Pi. While QEMU is useful for rapid prototyping of the Rust code, it can silently introduce incompatibilities at the library or API level if the two environments differ in version. For example, running a binary compiled for one version of OP-TEE on a mismatched runtime led to silent execution failures. To mitigate this, we standardized on a single version for both QEMU and Raspberry Pi. This required identifying and aligning OP-TEE releases manually and ultimately re-flashing the Raspberry Pi to match the QEMU build environment, adding a layer of maintenance overhead.

Lastly, OP-TEE on Raspberry Pi lacks native support for camera drivers, making it extremely complex to capture images directly within the secure environment. To overcome this limitation, we opted to use a secondary Raspberry Pi that runs a standard Linux distribution without OP-TEE. This device was configured from scratch, including user creation, SSH setup, and network configuration. A compatible camera was then identified, along with software capable of interfacing with it. This Pi captures the image and handles pre-processing (e.g., encryption), before transferring the data to the OP-TEE-enabled device for secure watermarking. While functional, this solution increased hardware complexity and the number of integration points in the system. This is also something that needs to be improved in the future so that only one edge device is used for the demo.

### 5.3 Semantic Integrity

Another major area of complexity was the evaluation of semantic integrity tools. Unlike other standardized tasks in computer vision or natural language processing, semantic integrity lacks established benchmarks and datasets. To address this, we conducted an in-depth literature review, which led us to discover the CASIA 2.0 dataset [54]. CASIA 2.0 is widely used in image forensics but only includes manipulated (positive) samples. This posed a challenge in evaluating model precision, as it lacked a natural control group of untouched image pairs. To mitigate this, we introduced a negative sampling strategy, generating image pairs that do not contain semantic manipulations, thus allowing us to measure false positives and better calibrate precision metrics.

The second significant challenge was the integration of diverse tools used for semantic integrity verification. These tools span multiple modalities and model types, including text

embeddings, visual embeddings, and visual-language models (VLMs). Each of these tools has different runtime requirements: some need GPU-accelerated servers, others run locally, and many expose their functionality through APIs. Managing these heterogeneous tools in a consistent and scalable way required careful planning and design. To solve this, we designed a set of abstract interfaces (*TextEmbeddingModel*, *VisualEmbeddingModel*, *VisualLLM*, ...), which standardize the interaction with different backends. We encapsulated model endpoints within Docker containers, allowing them to be deployed and invoked through a common HTTP-based interface. Finally, devising a common configuration file for all semantic integrity tools allowed to document the metadata for each model (e.g., endpoint URLs, required prompts, GPU constraints), ensuring clarity and reproducibility. For the final prediction layer, we adopted a simple majority voting mechanism to aggregate results from the different models. While effective as a first approach, we recognize that this component could benefit from further sophistication and plan to experiment with ensemble learning or confidence-based aggregation in future iterations.

## 6 Conclusion

In this report, we described Deepshield, a multi-layered framework for image authenticity and integrity verifications that combines cryptographic watermarking, Trusted Execution Environments (TEEs), and semantic integrity analysis. Our system targets the dual challenge of verifying content origin and detecting semantic manipulation, addressing threats that go beyond pixel-level forgery. The watermarking module operates in the frequency domain using a 2DWT-DCT embedding scheme, ensuring both imperceptibility and robustness against common distortions. The embedding and extraction pipeline is designed to run securely inside ARM-based TEEs using OP-TEE and Rust, making it suitable for deployment on resource-constrained edge devices.

Beyond watermarking, the system integrates a semantic integrity verification module composed of multiple complementary tools, including caption-based similarity, object detection, image embeddings, and ORB key-point matching. These tools capture different dimensions of semantic consistency and are aggregated through a modular voting mechanism.

Evaluation results confirm the effectiveness of our approach: watermarking proves robust across a wide range of distortions with minimal quality loss, while semantic integrity verification achieves high precision, particularly for splicing attacks. The integration of both modules yields a scalable and privacy-preserving verification pipeline that can operate across heterogeneous devices and attack surfaces.

### 6.1 Possible Extension and Future Work

While the current implementation of Deepshield demonstrates promising results in both watermarking and semantic integrity verification, several directions remain open for future improvement. One of the key limitations of our current watermarking approach is the possibility of approximating the original image through reverse engineering. To mitigate this, future work may explore the integration of homomorphic encryption and collaborative secret sharing [55, 56]. This would allow watermark embedding and verification to occur within the encrypted domain, ensuring that ownership claims can be validated in a confidential and privacy preserving fashion without revealing the original image or watermark.

Another promising direction involves improving the scalability of the verification pipeline. In large-scale deployments, narrowing the complexity of image verification is essential. To this end, a similarity-based retrieval step could be introduced, using visual embeddings (as already computed for semantic integrity) and a vector database such as FAISS [57]. This would allow the system to identify candidate matches quickly, reducing the need for exhaustive pairwise comparisons.

Additional future extensions could further improve the system’s precision and adaptability. Optimization of existing tools by identifying and tuning optimal parameters would enhance performance. Moreover, extending the image verification process to include not only watermarking and registering real images, but also tagging fake images generated by trusted tools, would increase the complexity of authenticity decisions while improving their accuracy.

Further improvements are also possible in the semantic integrity module. These include extending the toolset with metadata analysis (using e.g. EXIF tooling) and scene graph comparison, refining the aggregation strategy through weighted voting, and improving model behavior through targeted prompt engineering and hyperparameter tuning. These efforts aim to increase the system precision and recall, especially in challenging cases such as copy-move forgeries or subtle contextual edits.

## 7 References

- [1] H. Nguyen, F. Fang, J. Yamagishi, and I. Echizen, "Multi-task learning for detecting and segmenting manipulated facial images and videos," in *Proc. Biometrics Workshop (IWBF)*, 2019.
- [2] T. Karras, S. Laine, and T. Aila, "A Style-Based Generator Architecture for Generative Adversarial Networks," in *Proc. CVPR*, 2019.
- [3] C. Chesney and D. Citron, "Deep Fakes: A Looming Challenge for Privacy, Democracy, and National Security," *California Law Review*, vol. 107, 2019.
- [4] M. Westerlund, "The Emergence of Deepfake Technology: A Review," *Technology Innovation Management Review*, vol. 9, no. 11, pp. 39-52, 2019.
- [5] European Commission, "Tackling Online Disinformation," [Online]. Available: <https://digital-strategy.ec.europa.eu/en/policies/online-disinformation>
- [6] Sensity AI, "Deepfake Detection Platform," [Online]. Available: <https://sensity.ai> and <https://docs.sensity.ai>
- [7] Intel, "Intel FakeCatcher," [Online]. Available: <https://www.intel.com/content/www/us/en/research/trusted-media-deepfake-detection.html> 96% Accuracy Rate mentioned here: <https://newsroom.intel.com/opinion/unlocking-potential-generative-ai>
- [8] Reality Defender, "Real-Time Deepfake Detection," [Online]. Available: <https://www.realitydefender.com>
- [9] Y. Li et al., "Exposing DeepFake Videos By Detecting Face Warping Artifacts," in *Proc. CVPR Workshops*, 2019.
- [10] A. Rossler et al., "FaceForensics++: Learning to Detect Manipulated Facial Images," in *Proc. ICCV*, 2019.
- [11] HyperVerge, "AI-Powered Deepfake Detection," [Online]. Available: <https://hyperverge.co/use-cases/deepfake-detection>
- [12] Sentinel AI, "Defending Against Deepfakes and Information Warfare," [Online]. Available: <https://thesentinel.ai>
- [13] Resemble AI, "Neural Speech Watermarker," [Online]. Available: <https://www.resemble.ai/watermarker>
- [14] DeepBrain AI, "AI Human & Deepfake Detection," [Online]. Available: <https://www.deepbrainai.io/blog>
- [15] M. A. Memon, M. H. Shah, and A. A. Memon, "Imperceptible and Secure Blind Image Watermarking using Spread Spectrum Scheme with Adaptive Embedding Strength," *Proc. IEEE*, 2023.
- [16] A. M. Rezaei and M. M. Haidar, "A Novel and Efficient Blind Image Watermarking in Transform Domain," *Procedia Computer Science*, vol. 167, pp. 100–109, 2019.
- [17] A. J. Pathan and H. D. Joshi, "Encryption Based Image Watermarking Algorithm in 2DWT-DCT Domains," *Procedia Computer Science*, vol. 182, pp. 240–248, 2021.
- [18] J. Zhang, Y. Liu, and L. Zhang, "Robust Blind Image Watermarking Based on Interest Points and Deep Feature Extraction," *Journal of Information Security and Applications*, vol. 73, 2024.
- [19] B. K. Sheriff and A. A. Koomson, "A Cryptographic and Watermarking Encryption Technique for Securing and Authentication of Digital Images," *International Journal of Computer Applications*, vol. 111, no. 3, pp. 24–29, 2015.
- [20] Z. Yuan et al., "FreqMark: Frequency-Aware Neural Watermarking with Robustness Against Regeneration Attacks," *arXiv preprint arXiv:2410.20824*, 2024.

- [21] M. Gupta and Y. Wang, "Watermark Anything: Efficient and High-Fidelity Neural Watermarking at Scale," *arXiv preprint arXiv:241.hasan1.07231*, 2024.
- [22] Y. Wang et al., "WAVES: A Benchmark for Evaluating the Robustness of Image Watermarking Systems," *arXiv preprint arXiv:2401.08573v3*, 2024.
- [23] H. Li et al., "Robust Image Watermarking using Stable Diffusion," *arXiv preprint arXiv:2401.04247*, 2024.
- [24] H. H. Nguyen et al., "Multimedia Semantic Integrity Assessment Using Joint Embedding of Images and Text," *Proc. ACM MM*, 2017.
- [25] M. Shao et al., "AIRD: Adversarial Learning Framework for Image Repurposing Detection," *arXiv preprint arXiv:1903.00788*, 2019.
- [26] Y. Li et al., "Semantic-CD: Remote Sensing Image Semantic Change Detection towards Open-Vocabulary Setting," *arXiv preprint arXiv:2501.06808*, 2024.
- [27] L. Chen et al., "Semantic Change Detection with Hypermaps," *arXiv preprint arXiv:1604.07513*, 2016.
- [28] F. Zhang et al., "SeFi-CD: A Semantic First Change Detection Paradigm That Can Detect Any Change You Want," *Remote Sensing*, vol. 16, no. 21, 4109, 2024.
- [29] S. Liu et al., "Semantic Information Collaboration Network for Semantic Change Detection in Remote Sensing Images," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2024.
- [30] X. Wang et al., "Joint Spatio-Temporal Modeling for the Semantic Change Detection in Remote Sensing Images," *arXiv preprint arXiv:2212.05245*, 2022.
- [31] Q. Zhang et al., "A Multi-Task Consistency Enhancement Network for Semantic Change Detection in HR Remote Sensing Images," *Remote Sensing*, vol. 15, no. 21, 5106, 2023.
- [32] Confidential Computing Consortium, "Basics of Trusted Execution Environments (TEEs): The Heart of Confidential Computing," [Online]. Available: <https://confidential-computing.io>
- [33] Chair for IT Security, "Trusted Execution Environment (TEE) and Software Security," [Online]. Available: <https://cispa.de>
- [34] OP-TEE, "Trusted Applications," [Online]. Available: [https://optee.readthedocs.io/en/latest/building/trusted\\_applications.html](https://optee.readthedocs.io/en/latest/building/trusted_applications.html)
- [35] OP-TEE, "Building OP-TEE with Rust," [Online]. Available: [https://optee.readthedocs.io/en/latest/building/optee\\_with\\_rust.html](https://optee.readthedocs.io/en/latest/building/optee_with_rust.html)
- [36] Apache Teaclave, "Overview of OP-TEE Rust Examples," [Online]. Available: <https://teaclave.apache.org>
- [37] Rust Crates, "optee\_teec - Host API" and "optee\_utee - TA API," [Online]. Available: <https://docs.rs>
- [38] OP-TEE, "Running on Raspberry Pi 3," [Online]. Available: <https://optee.readthedocs.io/en/latest/architecture/rpi3.html>
- [39] OP-TEE, "QEMU Emulation Support," [Online]. Available: <https://optee.readthedocs.io/en/latest/building/devices/qemu.html>
- [40] OP-TEE FAQ, "Programming Languages Supported," [Online]. Available: <https://optee.readthedocs.io/en/latest/faq/faq.html>
- [41] M. Fan et al., "Darknetz: Privacy-Preserving Deep Neural Network Inference using ARM TrustZone," GitHub Repository: <https://github.com/mofanv/darknetz>
- [42] F. Tramèr and D. Boneh, "Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware," GitHub Repository: <https://github.com/ftramer/slalom>



- [43] OpenMined, "Secure Aggregation with PySyft, PyTorch and Intel SGX," [Online]. Available : <https://blog.openmined.org/secure-aggregation-on-trusted-execution-environments>
- [44] A. Jaiswal, R. Abd-Almageed, Y. Wu, and P. Natarajan, "AIRD: Adversarial Learning Framework for Image Repurposing Detection," *arXiv preprint*, arXiv:1903.00788, 2019.
- [45] E. Sabir, J. Cheng, A. Jaiswal, W. AbdAlmageed, and P. Natarajan, "Deep Multi-modal Image-Repurposing Detection," in *Proc. 26th ACM Int. Conf. on Multimedia (ACM MM)*, 2018, pp. 1337–1345. doi: 10.1145/3240508.3240707.
- [46] OP-TEE, "About OP-TEE," [Online]. Available: <https://optee.readthedocs.io/en/latest/general/about.html>.
- [47] Apache, "Teaclave TrustZone SDK Documentation," [Online]. Available: <https://github.com/apache/incubator-teaclave-trustzone-sdk#documentation>.
- [48] openSUSE, "QEMU Overview," [Online]. Available: <https://doc.opensuse.org/documentation/leap/virtualization/html/book-virtualization/cha-qemu-overview.html>.
- [49] Jolle Jolles, "The Raspberry Pi Guide," [Online]. Available: <https://raspberrypi-guide.github.io/raspberry-pi>.
- [50] OpenCV, "OpenCV: ORB (Oriented FAST and Rotated BRIEF)" [Online]. Available: [https://docs.opencv.org/4.x/d1/d89/tutorial\\_py\\_orb.html](https://docs.opencv.org/4.x/d1/d89/tutorial_py_orb.html)
- [51] Huggingface, "Qwen/Qwen2.5-VL-7B-Instruct". [Online]. Available: <https://huggingface.co/Qwen/Qwen2.5-VL-7B-Instruct>
- [52] Huggingface, "jinaai/jina-embeddings-v2-base-en". [Online]. Available: <https://huggingface.co/jinaai/jina-embeddings-v2-base-en>
- [53] Huggingface, "facebook/dinov2-base". [Online]. Available: <https://huggingface.co/facebook/dinov2-base>
- [54] CASIA v2.0 Image Tampering Detection Evaluation Database v2.0 [Online]. Available: <https://github.com/namtpham/casia2groundtruth?tab=readme-ov-file>
- [55] Gentry, C. (2009). *A Fully Homomorphic Encryption Scheme* (Doctoral dissertation, Stanford University). Stanford University. <https://crypto.stanford.edu/craig>
- [56] Chen, H., Laine, K., & Player, R. (2017). *Simple Encrypted Arithmetic Library – SEAL v2.1*. Microsoft Research. <https://www.microsoft.com/en-us/research/project/microsoft-seal/>

[57] Johnson, J., Douze, M., & Jégou, H. (2019). *Billion-scale similarity search with GPUs*. *IEEE Transactions on Big Data*, 7(3), 535–547.  
<https://doi.org/10.1109/TBDATA.2019.2921572>