Deep Learning-Based Attacks on Traditional Watermarking Systems in Real-Time Live Video Streams

Huixin Wang[1], Amin Sakzad[1], Stuart Hall[1]
[1]Faculty of Information Technology, Monash University
hwan0158@student.monash.edu, amin.sakzad, stuart.hall@monash.edu

***Abstract*—Traditional explicit and implicit watermarking methods have long been employed to protect multimedia content. However, while implicit watermarks offer robustness, integrating them into real-time streaming environments introduces technical challenges, such as increased latency and high computational overhead. As a result, major platforms like YouTube and broadcast channels continue to rely on visible, image-based watermarks, reasoning that removing such marks in real-time is non-trivial. In image processing, deep learning-based inpainting techniques have been used to remove visible watermarks. Yet, traditional supervised convolutional neural networks (CNNs) often depend on paired training data, which may not be available in practice. Chunwei Tian et al. proposed the Self-Supervised CNN (SWCNN) [8], which circumvents this dependency by generating its own reference images. Through a lightweight heterogeneous U-Net architecture, SWCNN effectively adapts to various watermark distributions, maintaining precision while reducing computational costs. Despite these advances, the original SWCNN implementation still struggles to meet the speed and latency demands of real-time video watermark removal, particularly for live streaming. To address this limitation, we introduce half-precision (FP16) computation, preprocessing strategies, and multi-threaded optimizations to SWCNN. Our enhanced approach achieves real-time performance not only for locally streamed video via VLC media player but also for live internet sources, such as Twitch streams. Experimental results confirm that our optimized SWCNN improves both watermark removal quality and frame rates across different resolutions, making it practical for real-time deployment. To support ongoing research and ensure reproducibility, we have created a new dataset of watermarked video frames. This work not only advances the field of video watermark removal but also exposes potential vulnerabilities in current watermarking strategies, reinforcing the need for stronger copyright protection measures. The code and all video demonstrations are available in the Supplemental material.**

***Index Terms*—Watermark removal, real-time video streaming, deep learning, SWCNN, half-precision computation.**

I. Introduction

Digital watermarking has become a standard practice for protecting the ownership of images and videos, enabling content creators and distributors to assert copyright and ensure traceability. As live streaming continues to gain prominence as a primary medium for media consumption, traditional frequency-domain watermarking techniques face increasing difficulties in meeting real-time performance demands [1]. These challenges arise from the inherent complexity of live video architectures and the delicate balance between video quality and transmission efficiency. For instance, varying Group of Pictures (GOP) structures, real-time bit-rate fluctuations, and metadata inaccuracies can all complicate the synchronization and stable embedding of watermarks in live streams [2]. Additionally, watermark processing often requires full decoding and re-encoding, increasing latency and the risk of visual artifacts, which further degrade the viewing experience.

In response, major platforms like YouTube recommend using simple, visible image-based watermarks that appear as small logos to indicate content ownership [3]. While this approach simplifies embedding, the real-time removal of such explicit watermarks remains a non-trivial technical challenge, especially for high-resolution content and varying watermark shapes and opacities [4]. Nonetheless, emerging deepfake technologies, like Deep-Live-Cam [5], demonstrate that real-time content manipulation is increasingly feasible with minimal inputs. As such methods evolve, there is a growing threat that unauthorized parties may remove watermarks from legitimate live streams, misleading audiences and enabling piracy and unlicensed rebroadcasting. Such deceptive practices can lead to significant financial loss, reputational damage for rights holders, and erosion of trust in live broadcasts [6].

In recent years, deep learning-based approaches have improved watermark removal techniques, particularly for still images. For example, WDNet [7] and the Self-Supervised Convolutional Neural Network (SWCNN) [8] have made notable progress. However, these methods struggle to achieve real-time performance in dynamic video streams. Existing techniques often depend on computationally expensive operations and paired datasets, limiting their applicability when working with unpredictable watermark patterns or rapidly changing live content. Achieving efficient, robust watermark removal during a live broadcast is further complicated by latency, bandwidth constraints, and the necessity of maintaining video quality.

Addressing this gap, our work proposes a real-time video watermark removal attack that not only performs effectively on locally stored videos but also extends seamlessly to genuine live-streaming environments, including those sourced from platforms like Twitch. By enabling this functionality, we aim to provide a practical solution that can be directly integrated

into real-world scenarios, ultimately aiding in copyright protection and mitigating unauthorized rebroadcasting. Beyond contributing technically to image restoration and inpainting research, our approach highlights critical weaknesses in current watermarking strategies and informs the development of more secure defense mechanisms. This study makes the following key contributions:

1) **Real-time optimized SWCNN**: We introduce half-precision computation, various pre-processing strategies, and multi-threaded optimization to substantially improve SWCNN's processing speed and efficiency. As a result, our optimized SWCNN achieves real-time frame rates (e.g., 65 fps at 320p and 6 fps at 1080p), meeting the demands of both on-premise setups and remote streaming sources.
2) **Robust watermark removal without paired data**: Leveraging SWCNN's self-supervised capabilities, we refined parameter tuning and training strategies to ensure accurate, artifact-free watermark removal, even in the absence of clean reference images or annotated data.
3) **Integrated real-time inference framework**: We developed a coherent pipeline that streamlines data loading, frame buffering, and inference, making it feasible to deploy our optimized SWCNN in real-time scenarios. Crucially, we validate its performance not only on local test streams but also on live content from platforms like Twitch.
4) **New dataset for live scenarios**: To support ongoing research, we curated a new dataset of watermarked video frames derived from popular video-on-demand platform logos. By simulating authentic live scenarios, this dataset aids in training and evaluating watermark removal attacks under realistic conditions.

The remainder of this paper is organized as follows: Section II reviews current watermark removal technologies, detailing the obstacles posed by real-time video applications and existing model limitations. Section III presents our optimization strategies to SWCNN. Section IV describes the experimental setup, including our dataset, hardware, and evaluation protocols. In Section V, we present and analyze experimental results, demonstrating the effectiveness of our real-time watermark removal attack on both local test streams and actual live Twitch feeds. Finally, Section VI summarizes the main contributions and discusses future research directions.

II. Background

In what follows we give a background on various watermarking techniques in live-streaming environments and their limitations.

A. Image Watermark Removal Attacks

Image inpainting [35] has served as a fundamental approach to watermark removal attacks, aiming to reconstruct missing or corrupted regions and preserve visual continuity. Traditional methods often rely on diffusion-based algorithms that propagate pixel information from known regions [36], or exemplar-based techniques that sample patches to fill occluded areas [37]. While effective in controlled scenarios, these classic approaches generally demand manual watermark localization and struggle with complex patterns, colors, and transparency levels, making them unsuitable for large-scale, real-time applications.

Deep learning has led to more advanced solutions, leveraging CNNs and GANs [9] to automate removal without manual intervention. Early methods like Context Encoders and Pix2Pix [38] utilized paired datasets for training, but such data can be difficult to obtain. In addition, GAN-based models can introduce artifacts and require careful regularization. Temporal consistency further complicates video watermark removal. Techniques incorporating optical flow and Temporal Convolutional Networks (TCNs) [10] enhance stability across frames, yet dynamic watermarks that change in position, shape, or appearance remain challenging. Achieving robust performance in complex, real-world conditions requires methods that can adapt without relying solely on paired training data or extensive annotations.

Over time, research has integrated multi-scale learning, data augmentation, attention mechanisms, and self-supervised techniques [11]–[16] to improve adaptability and reduce data dependence. However, while these innovations offer strong results on offline datasets, scaling them to real-time streaming contexts introduces new obstacles.

B. Challenges in Live Streaming Environments

Adapting watermark removal to live streaming scenarios involves overcoming hurdles distinct from offline processing. Modern streaming protocols, such as HTTP Live Streaming (HLS), deliver content in segments with dynamically changing bitrates and Group of Pictures (GOP) structures. These variations make it difficult to maintain synchronization between the watermark removal model and the incoming stream. Real-time fluctuations, network jitter, and adaptive bitrate changes demand that watermark removal be both time-sensitive and robust to non-uniform video quality. Integrating with live stream players (e.g., VLC or web-based clients) adds additional complexity, as frames must be processed with minimal latency, and any delay can lead to unsmooth playback or buffering. Compared to offline datasets, live-streams lack consistent ground truths and can include watermarks that appear, disappear, or shift unpredictably. The inability to pause or reprocess frames necessitates efficient inference with limited memory and computational overhead. In addition, any attack model must handle a broad spectrum of watermark types and dynamic backgrounds without introducing visible artifacts or degrading viewer experience.

C. Existing Attacks and Their Limitations in Real-Time Streaming

Recent deep learning-based attacks, such as WDNet [7] and SWCNN [16], have shown marked improvements in

watermark removal. WDNet employs a two-stage approach using large CNN backbones, achieving high-quality results on benchmark datasets but suffering from high computational costs and slow inference, which are impractical in streaming conditions. SWCNN introduced a self-supervised approach with a heterogeneous U-Net architecture to reduce dependency on paired datasets and enhance efficiency. Its lightweight design and hybrid loss function improve adaptability and visual quality compared to more computationally heavy models. Nevertheless, in its native form, SWCNN's speed and resource demands do not meet the stringent requirements of live streaming. Its reported performance — about 20 fps at lower resolutions — cannot cope with high-resolution, time-sensitive contexts, making it inadequate for handling real-time changes in bitrate, synchronization with HLS segments, or seamless player integration.

While other optimization strategies, such as half-precision computation, multi-threading, quantization, pruning, and efficient network architectures, have been explored in related domains [17]–[26], few have been systematically applied to video watermark removal attacks in live streaming. Existing solutions generally focus on offline processing or do not reliably scale to fluctuating streaming conditions. This gap highlights the need for a specifically optimized approach that addresses both the speed and adaptability demanded by real-time broadcasts. By contrasting our work with previous methods that struggle in dynamic, network-driven streaming environments, we highlight this paper's innovation: a real-time watermark removal pipeline that integrates SWCNN with targeted optimizations. Our approach accommodates HLS-based streams, adapts to varying resolutions and bitrates, and ensures temporal consistency and low-latency inference. In doing so, it opens up new possibilities for copyright protection, watermark resilience, and content integrity within live streaming scenarios.

D. Threat Model

In what follows, we outline our threst model.

a) Adversary and Goal: We consider an unauthorized endpoint adversary who aims to remove *visible* overlay watermarks from live video in real time while preserving perceptual quality and continuous playback for immediate viewing or redistribution.

b) Capabilities: The adversary can access *decoded* frames at the endpoint, buffer a short sliding window, and run a local preprocessing→inference→postprocessing pipeline on commodity hardware in parallel with playback. The adversary does not control the broadcaster, CDN, or encoder, and cannot rewind beyond the player's short buffer.

c) Knowledge: The ground-truth watermark layer and its generation process are unknown. Models are trained offline with *synthetic* overlays and applied zero-shot to live content; online processing is limited to lightweight localization/normalization. Watermarks may vary over time in position, opacity, or shape; backgrounds are complex and dynamic.

d) Constraints and Success: Segmented adaptive streaming, GOP structure, and network jitter impose tight latency and throughput budgets. The attack must meet per-frame deadlines without introducing stalls, tolerate resolution/bitrate fluctuations, and operate under bounded compute and memory. Success is defined by stall-free playback, low perceptual residuals/artifacts with temporal coherence, and scenario-appropriate throughput. When reference data exists we report PSNR/SSIM; otherwise we use no-reference indicators and human inspection.

e) Scope and Non-Goals: We target endpoint-visible overlays in decoded video. Out of scope are DRM-protected playback paths that block frame access, attacks requiring control of the source or backend, invisible/forensic watermarks at the codec/signal level, and cryptographically verifiable or signed provenance mechanisms, which change the trust model and are orthogonal to endpoint-side removal.

f) Defender Perspective: This model captures an endpoint re-streaming threat and motivates hardening via motion-aware and placement-randomized overlays, content-adaptive opacity, and, where feasible, verifiable provenance to complement visible marks.

III. Our Attack

To enable real-time watermark removal suitable for live streaming scenarios, we build upon SWCNN's self-supervised and lightweight architecture. Rather than relying heavily on paired training data, SWCNN leverages random watermark generation to learn robust removal strategies. We maintain these strengths while introducing specialized optimizations such as half-precision inference, multi-threaded pipeline design, and refined pre-processing to achieve high frame rates and efficient handling of dynamic streaming conditions. Our modifications streamline data handling and reduce computational overhead, making SWCNN more practical for real-time deployments. This includes efficient frame buffering, adaptive scaling strategies for diverse resolutions, and hardware-level enhancements that enable our model to operate under tight latency constraints. The following sections detail how we incorporate these optimizations.

A. Our optimization techniques

The original SWCNN was developed for static image watermark removal. In this work, we extended the SWCNN architecture to handle video inputs, enabling real-time watermark removal from video streams.

1) SWCNN for Video Inference: We adapt SWCNN with careful considerations on both computational efficiency and maintaining high watermark removal quality. We implemented a video inference pipeline using the SWCNN model, adding features such as real-time frame rate display (FPS) to facilitate practical evaluation. **Algorithm 1** outlines the video inference process using the SWCNN model:

The red-highlighted sections indicate the author's innovations, including fast preprocessing, FP16 inference, and cross-frame temporal smoothing techniques to enhance the performance of real-time video watermark removal.

Initialize SWCNN model with pre-trained weights.
Open video file using OpenCV's `VideoCapture`.
Start time for FPS calculation to evaluate the real-time capability of the model.
while video frames are available **do**
  Read a frame from the video.
  Preprocess the frame:
    1) Normalize pixel values to [0, 1].
    2) Resize if necessary to meet input size requirements.
    3) Convert to tensor and move to GPU.
    4) Apply fast downsampling or denoising to improve runtime efficiency.
  Perform Inference:
    1) Pass the preprocessed frame through the SWCNN model.
    2) Clamp output values to [0, 1].
    3) Use half-precision (FP16) inference to reduce computational overhead.
  Postprocess the output:
    1) Convert tensor to NumPy array.
    2) Scale pixel values to [0, 255] for display.
    3) Optionally apply temporal smoothing across frames to reduce flicker.
  Display original and processed frames side by side.
end while
Release video capture and close display windows.

2) Integrating with Local and Online Streaming Sources: To enable our attack and support real-time watermark removal in practical scenarios, we integrated our optimized SWCNN into pipelines for both local and online live streams.

For local streams, we employed VLC-based setups, where videos are served locally and accessed through low-latency interfaces. Using `OpenCV`'s `VideoCapture` with appropriate backends, we ensured minimal overhead in frame acquisition.

For online sources, such as Twitch, we utilized m3u8 playlists to capture live HTTP Live Streaming (HLS) video segments. This introduces varying bitrates, GOP structures, and network jitter. Frames must be extracted and processed on-the-fly, maintaining stable performance despite unpredictable conditions. By handling both local and remote inputs through a unified pre-processing pipeline consisting of normalization, re-sizing, and tensor conversion. We also enabled seamless adaptation between stable local feeds and dynamic internet streams without additional modifications.

3) Half-Precision Computation (FP16): To reduce computational load and memory consumption during video processing, we employed half-precision (FP16) calculations for neural network inference. Specifically, we used PyTorch's mixed-precision module `torch.cuda.amp.autocast` to perform computations in FP16 where appropriate. This approach significantly decreases GPU memory usage and accelerates computation, making it suitable for real-time applications. Prior research indicates that FP16 inference can improve throughput by approximately 30% without notable accuracy loss [17]. By leveraging FP16, our attack processes incoming frames more quickly, a crucial factor in handling high-resolution content or adapting to fluctuating bitrates in HLS streams.

4) Preprocessing Techniques: We employ various efficient pre-processing techniques to maximizes model performance during inference:

- **Frame Resizing**: We standardize the input frame size (e.g., 640×360) to ensure consistent computational demands. This prevents resolution variability from causing slowdowns.
- **Normalization**: We scale all pixel values to be inside the interval [0, 1]. This ensures stable input distributions for the network.
- **Tensor Conversion**: We convert frames into FP16 tensors for seamless integration with the mixed-precision inference pipeline.

Furthermore, the use of `.permute(2, 0, 1)` rearranges the dimensions to match the expected input format for PyTorch models (channel-first). These steps yield uniform input conditions, facilitating smooth adaptation to both local and online streams, regardless of their original resolution or encoding complexity.

5) Multithreading, Code Optimization, and Dynamicity: By distributing tasks among separate threads and using a queue-based producer-consumer model, we prevent any single component such as frame reading or model inference from becoming a performance bottleneck. Batch processing of frames (e.g., `batch_size=2`) further exploits GPU parallelism, allowing multiple frames to be processed simultaneously. This batching not only improves GPU utilization but also helps absorb momentary fluctuations in frame arrival times from online streams. Even if the network stalls briefly, the inference thread can process buffered frames, maintaining a stable output rate.

Our approach allows the system to adapt dynamically: if the online stream's quality drops or latency spikes, the pipeline can continue processing available frames without interruption. Conversely, when conditions improve, the model's optimized performance ensures that higher-resolution frames can be handled smoothly, maintaining visual clarity and consistent watermark removal effectiveness.

6) Training Enhancements: To improve model performance and stability, we introduced additional training strategies:

- **Data Augmentation**: We implement random cropping, flipping, and color jitter to increase data diversity. This results in improving generalization to unseen watermarks.
- **Dynamic Learning Rate Adjustment**: We also adjust the learning rate during training to prevent overfitting. It further helps the model to converge to a robust solution.

- **Efficient Data Loading**: We employ multiple workers and parallel data loaders to expedite the data retrieval process. This also enables using of larger batch sizes and smoother training dynamics.

B. Training Data Generation and Preparation

Following the original SWCNN approach [16], we use synthetic watermarks generated from multiple brand logos. To enhance the model's applicability to global video-on-demand and multimedia platforms, we selected **16 common international video-on-demand platform logos and film IP logos** from an open-source Kaggle logo dataset [29], please see Fig. 1. This dataset contains vector images of some popular brand logos, totalling 1,481 international popular brand logos and files. To prepare the logos for watermarking,

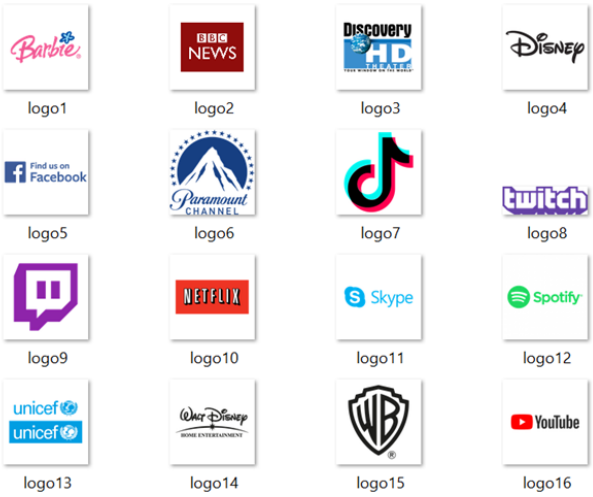

Fig. 1: A collection of 16 international popular brand logos.

we performed transparent background processing. Using the `load_dataset` function from the `datasets` library, we loaded the logo dataset and processed each image by converting white pixels to transparent pixels. The processed images were then saved in PNG format with transparent backgrounds, making them suitable for overlaying onto video frames as watermarks.

For the training images, we diverged from the common practice of using standard image datasets like PASCAL VOC 2012. Instead, we selected a video dataset from Hugging Face [28], specifically the `test_raw_video_data` dataset. From this dataset, we randomly selected **100 videos** and extracted frames via `ffmpeg`, saving them in JPEG compressed format. The frame extraction process depended on the video's frame rate:

- **For videos at or below 30 frames per second (fps):** We extracted all frames.
- **For videos at 60 fps:** We extracted every other frame to reduce the total number of frames.

During extraction, we ensured that the dimensions of each frame were multiples of 32 by setting the width and height to setting the width and height to `(int(w/32) * 32)` and `(int(h/32) * 32)`, where w and h are the original width and height, respectively. This step is crucial to prevent errors during training caused by incompatible input sizes, as many CNNs require the input dimensions to be divisible by the downsampling factors used in the network.

Each video frame in the training dataset was randomly assigned a logo from the processed logo dataset to serve as a watermark. The watermarks were applied with transparency levels of **0.3**, **0.5**, **0.7**, and **1.0** to simulate varying degrees of visibility. The coverage of the watermark varied in a self-supervised manner from **0** to **0.4**, with the watermark size set between **0.5 to 1 times** the original size.

To accelerate training and manage computational resources, we cropped each watermarked image into patches of **256×256 pixels**. This resulted in a total of **3,111 patches per image**, providing a large and diverse set of training samples. Cropping into smaller patches not only reduces memory consumption but also increases the effective batch size, facilitating more stable and efficient training.

IV. Experimental Setup

We now explain the setup for our experiments. The code for our evaluation is available at [39].

A. Hardware and Software Environment

All experiments were conducted on systems equipped with AMD Ryzen 9 7000 Series CPUs and NVIDIA RTX 4070 GPU running Windows, using Python 3.9 along with the dependencies listed in Table I. The CUDA Toolkit 11.3 and cuDNN 8.0.5 enabled GPU acceleration, significantly reducing training times by leveraging the parallel processing capabilities of the GPUs. Additional libraries such as scikit-image and OpenCV-Python were used for image processing tasks, including data augmentation and preprocessing steps. TensorBoard was employed for monitoring the training process, providing visualizations of metrics such as loss and accuracy over epochs. Matplotlib was used for plotting and visualizing results, and FFmpeg facilitated video processing tasks, including frame extraction and video reconstruction. For extensive model training and optimization, we chose the

TABLE I: Software Dependencies and Versions

Library	Version
torchvision	0.12.0
torchaudio	0.11.0
cudatoolkit	11.3
tensorboard	2.9.1
scikit-image	0.19.3
opencv-python	4.6.0
matplotlib	3.5.2
ffmpeg	4.2.3
CUDA	10.2
cuDNN	8.0.5

AutoDL platform. The configuration details of the AutoDL platform are presented in Table II.

TABLE II: AutoDL Platform Configuration

Component	Details
Operating System	Ubuntu 20.04
GPU	NVIDIA RTX 4090D (24 GB VRAM)
CPU	18 vCPU AMD EPYC 9754 128-Core Processor
Memory	60 GB
Disk (System)	30 GB
Disk (Data)	50 GB

B. Training Configuration

To optimize the SWCNN model for real-time video watermark removal attack, we carefully configured the training parameters and implementation details, ensuring efficient learning and generalization to various watermark types.

1) Training Parameters: We made several modifications to the original SWCNN training parameters to enhance the model's performance and training efficiency. The **batch size** was increased from 4 to 8, allowing the model to process more samples per iteration and potentially speeding up the training process. This change leverages the available GPU memory more effectively, provided that the hardware resources are sufficient to handle the increased load.

The **number of epochs** was set to 300 to give the model ample time to learn and converge. A longer training duration helps the model to better generalize by thoroughly exploring the parameter space. The **learning rate** was initialized at 1×10^{-3} and scheduled to decay by a factor of 0.1 after every 30 epochs. This step decay strategy allows for larger updates in the initial stages of training and finer adjustments as the training progresses, facilitating convergence to a better minimum.

We used the **Adam optimizer** for its efficiency and adaptive learning rate capabilities, keeping the default parameters $\beta_1 = 0.9$ and $\beta_2 = 0.999$. The Adam optimizer effectively handles sparse gradients and adjusts learning rates on a per-parameter basis, which is beneficial for the complex optimization landscape of deep neural networks.

Additionally, we increased the **watermark opacity** (α) from 0.3 to 0.5, making the watermark more prominent and the removal task more challenging. This adjustment aims to improve the model's robustness by training it on more difficult examples, thereby enhancing its ability to handle a wider range of watermark intensities during inference.

We chose the L1 loss because it is less sensitive to outliers and encourages sparsity, which leads to better preservation of edges and fine details. This decision aligns with findings from prior research, indicating that the L1 loss is particularly suitable for image watermark removal using SWCNN. According to quantitative and qualitative analyses presented in the original work [8], the L1 loss effectively guides the model to produce high-quality, watermark-free images without the need for additional loss components.

We integrated **TensorBoard** logging into our training pipeline for real-time monitoring and qualitative assessment of the model's performance. After each epoch, sample images from the validation set including the original, watermarked, and reconstructed images were logged (see Fig. 2). This visual feedback allowed us to observe the attack's progress in removing watermarks and preserving image details over time. We updated the validation loop by employing the

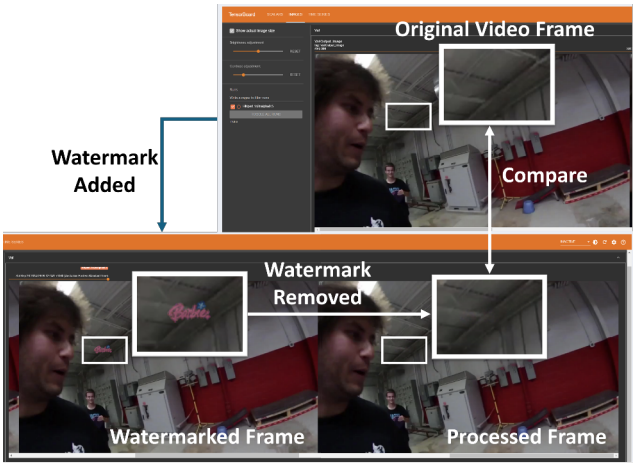

Fig. 2: TensorBoard screenshot showing the original, watermarked, and processed frames for watermark removal evaluation.

`torch.no_grad()` context manager to wrap the validation steps, which disables gradient computations during validation. This approach reduced memory usage and computational overhead.

2) Validation Strategy: For validation, we prepared a separate set of images processed to ensure their dimensions were multiples of 32, matching the network's requirements. Watermarks were applied to these images with the same opacity and augmentation settings used during training to maintain consistency. We evaluated the model's performance by computing the **Peak Signal-to-Noise Ratio (PSNR)** between the reconstructed images and the ground truth. Logging validation images to TensorBoard provided qualitative insights into the model's effectiveness in removing watermarks and preserving image quality.

3) Training Workflow: The training workflow can be summarized as follows: **Data Loading**: The training and validation datasets were loaded using PyTorch's DataLoader, with multi-threading enabled (`num_workers=8`) to accelerate data loading.

Model Initialization: The SWCNN model was initialized, and if multiple GPUs were available, parallelization was applied.

Optimization Loop:

- For each epoch:
  - Adjust the learning rate if necessary.
  - For each batch, we: (1) apply data augmentation and watermark blending to generate input images, (2) perform a forward pass to obtain model outputs, (3) compute the total loss using the L1 loss function,

and (4) perform backpropagation and update model parameters using the optimizer.
  - At the end of the epoch, we: (1) evaluate the model on the validation set, (2) log metrics and sample outputs to TensorBoard, and (3) save the model checkpoint.

C. Evaluation Metrics and Comparative Methods

To validate the effectiveness of our optimized SWCNN model, we compared it with the unoptimized version and different optimized variants incorporating various techniques. The models evaluated include the **original SWCNN** (baseline), **SWCNN with FP16 computation**, **SWCNN with preprocessing optimizations (PPFP16)**, and **SWCNN with multithreading optimization**. We employed the following evaluation metrics to provide a balanced assessment of both qualitative and quantitative aspects of watermark removal performance and model efficiency:

- **Peak Signal-to-Noise Ratio (PSNR)**: It measures the similarity between the reconstructed (watermark-removed) image and the original clean image. The higher PSNR values indicate better image quality. The script leverages FFmpeg's built-in filters to perform these computations efficiently. For PSNR calculation, the script compares each processed video with the original using FFmpeg's PSNR filter, extracts the average PSNR value from the output, and records it. This automated approach, that we also use for the next performance measure calculation as well, ensures consistency and accuracy in evaluating the models' performance.
- **Structural Similarity Index Measure (SSIM)**: It assesses the perceptual similarity between the reconstructed image and the original image, considering luminance, contrast, and structure. SSIM values range from 0 to 1, with higher values indicating greater similarity. We developed a Python script to automate the calculation of SSIM between the original and processed videos. For SSIM calculation, the script employs FFmpeg's SSIM filter to generate a log file containing detailed SSIM statistics, from which it extracts the overall SSIM value for each video comparison.
- **Frames Per Second (FPS)**: It indicates the real-time performance of the model during video inference, showing how many frames the model can process per second. Higher FPS values signify better efficiency.
- **Computational Efficiency**: Evaluated by measuring inference time per frame, GPU memory usage, and CPU utilization. This assessment helps determine the attack's practicality for deployment in resource-constrained environments.

By comparing these models using the above metrics, we aimed to demonstrate how each optimization contributes to performance improvements in real-time video watermark removal attack.

D. Deployment in Realistic Streaming Scenarios

Beyond static inputs, our approach readily adapts to both locally served streams and online platforms. For local setups, VLC's "Media → Stream" functionality and an OpenCV `VideoCapture` interface enable low-latency frame acquisition from a dedicated HTTP endpoint. Frames are normalized, resized, and converted into tensors before undergoing Optimised-SWCNN inference, ensuring high frame rates under controlled conditions.

For online sources (e.g., Twitch), extracting the m3u8 playlist URL via browser tools allows HLS segments to be processed on-the-fly as shown in Fig. 3 Despite varying bitrates, GOP structures, and intermittent network fluctuations, uniform preprocessing and half-precision inference maintain stable, real-time watermark removal. The pipeline thus adapts seamlessly to evolving input characteristics, preserving visual quality and low latency in diverse streaming environments.

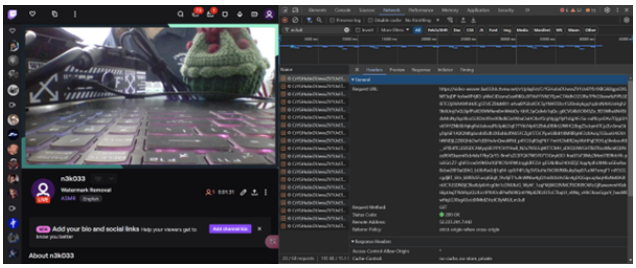

Fig. 3: Demonstration of extracting an m3u8 playlist URL using browser developer tools.

E. Ethical Considerations

This study was conducted using publicly available, open-source datasets intended solely for research purposes. The datasets do not contain any personal or sensitive information that could identify individuals. All experiments and data handling procedures complied with ethical standards and guidelines relevant to research in this field. As such, no additional ethical approval was required for this work.

V. Results and Discussion

In this section, we present a comprehensive evaluation of our optimized SWCNN models for real-time video watermark removal attack. Our analysis focuses on both the effectiveness of watermark removal and the computational efficiency of each attack variant. We also discuss the trade-offs involved in our optimization strategies.

A. Training Results

Our optimized SWCNN model underwent extensive training over 300 epochs. By the end of training, it achieved a PSNR of 59.21 dB on the validation dataset, significantly outperforming the original model at an opacity level of $\alpha = 0.5$ (see Table III). This high PSNR indicates excellent image quality in watermark removal tasks. Table III furrehr illustrates the performance of different methods for removing watermarks

TABLE III: PSNR Comparison at Opacity $\alpha = 0.5$

Method	PSNR (dB)
DnCNN [26]	29.0817
FFDNet [31]	26.6991
DRD-Net [32]	29.5227
FastDerainNet [33]	29.0564
EAFNWDD [34]	32.6197
SWCNN [8]	33.6491
Optimised SWCNN (Ours)	**59.2143**

at a transparency level of 0.5. Fig. 4 shows the progression of PSNR on both the training and validation sets throughout the epochs. The consistent upward trend and convergence demonstrate the model's robust learning and its ability to generalize well to unseen data.

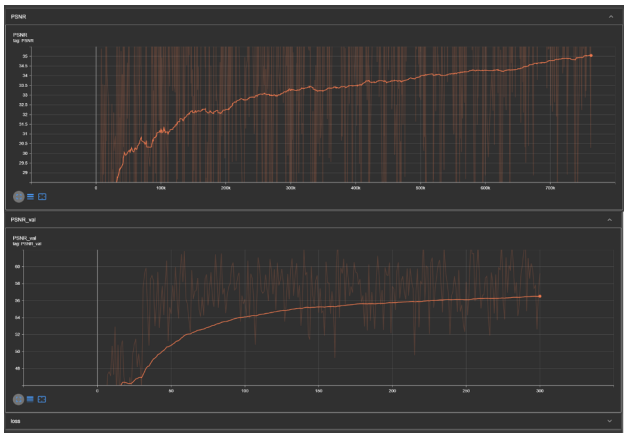

Fig. 4: Training and validation PSNR progression across epochs.

B. Evaluation of Watermark Removal Performance

We now discuss the performance of our watermark removal attack both quantitatively and qualitatively.

1) Quantitative Evaluation using PSNR and SSIM: We now test our attack consisted of videos at a resolution of 320p, each embedded with watermarks at different opacity levels $\alpha = 0.3, 0.5, 0.7$, and 1.0. The original, unwatermarked videos served as the ground truth for comparison. Table IV presents the PSNR and SSIM values for each model across different watermark opacities.

The quantitative results indicate that the SWCNN with FP16 Computation consistently achieved the highest PSNR and SSIM values, slightly outperforming the unoptimized model [8]. This suggests that half-precision computation not only improves computational efficiency but also enhances image quality. The PPFP16 and Multithreading models showed lower PSNR and SSIM values, implying that while these optimizations speed up processing, they may compromise image quality. As expected, all models experienced a decrease in PSNR and SSIM as the watermark opacity increased, due to the greater challenge in removing more opaque watermarks.

2) Visual Quality Assessment: To visually evaluate the effectiveness of our optimized models, we conducted comparative experiments using the same four versions of the SWCNN model. Representative frames from the test set containing watermarks at an opacity level of $\alpha = 0.5$ and a resolution of 320p were selected. Figures 5–8 present side-by-side comparisons of the original watermarked frames and the outputs from each model variant. This visual assessment allowed us to qualitatively analyze the effectiveness of watermark removal and any potential impact on image quality introduced by the optimizations.

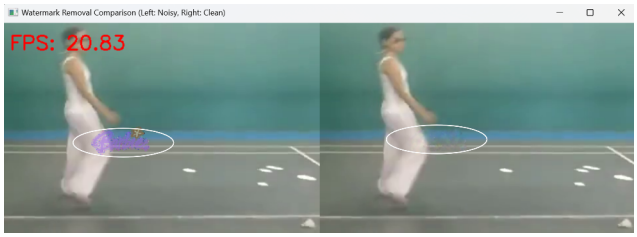

Fig. 5: Visual Comparison of Watermark Removal - Unoptimized SWCNN (FPS: 27.76, PSNR: 42.75 dB, SSIM: 0.9934).

Fig. 6: Visual Comparison of Watermark Removal - SWCNN with FP16 Computation (FPS: 32.53, PSNR: 43.17 dB, SSIM: 0.9932).

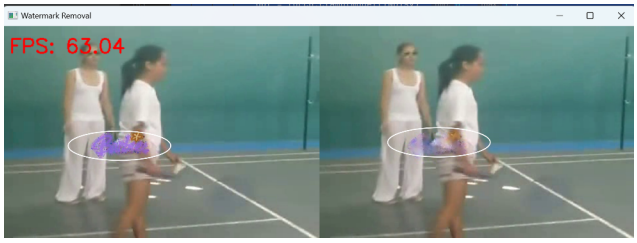

Fig. 7: Visual Comparison of Watermark Removal - SWCNN with PPFP16 (FPS: 64.64, PSNR: 37.83 dB, SSIM: 0.9862).

The visual comparisons reveal the following insights:

- *Unoptimized SWCNN [8]:* effectively removes the watermark with high image quality but has lower processing speed, resulting in less smooth playback.
- *SWCNN with FP16 Computation:* maintains excellent image quality with slightly improved processing speed, indicating a successful optimization without compromising visual fidelity.

TABLE IV: PSNR and SSIM at Different Watermark Opacities α.

Model	PSNR ($\alpha = 0.3$)		SSIM ($\alpha = 0.3$)		PSNR ($\alpha = 0.5$)		SSIM ($\alpha = 0.5$)	
Unoptimized SWCNN [8]	36.77 dB	0.9862	42.75 dB	0.9934	36.72 dB	0.9790	34.14 dB	0.9848
SWCNN with FP16 Computation	36.80 dB	0.9862	43.17 dB	0.9932	36.79 dB	0.9790	34.16 dB	0.9848
SWCNN with PPFP16	33.04 dB	0.9786	37.83 dB	0.9862	35.04 dB	0.9778	34.61 dB	0.9783
SWCNN with Multithreading	31.82 dB	0.9786	34.87 dB	0.9862	33.25 dB	0.9778	32.95 dB	0.9783

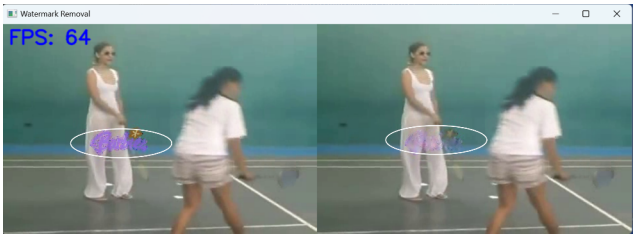

Fig. 8: Visual Comparison of Watermark Removal - SWCNN with Multithreading (FPS: 66.55, PSNR: 34.87 dB, SSIM: 0.9862).

- *SWCNN with PPFP16 and Multithreading Optimization:* achieves higher frames per second (FPS), resulting in smoother playback. However, there is a noticeable degradation in image quality, with residual watermark artifacts present. This suggests that while these optimizations enhance efficiency, they may negatively impact the effectiveness of watermark removal.

The visual observations are supported by the quantitative metrics from the PSNR and SSIM calculations, providing empirical evidence of the performance differences among the models.

The experiments demonstrated that the SWCNN with FP16 Computation offers the best balance between image quality and computational efficiency. While the PPFP16 and Multithreading optimizations significantly improve processing speed, they may introduce quality degradation that could be unacceptable in applications where visual fidelity is critical. Our evaluation highlights the importance of considering both quantitative and qualitative factors when assessing model optimizations for real-time video watermark removal.

C. Computational Efficiency Analysis

In this section, we evaluate the computational efficiency of our optimized models across different video resolutions to determine their real-time processing capabilities. We assessed performance metrics such as frames per second (FPS), average CPU utilization, and average GPU utilization at video resolutions of 320p, 480p, 720p, and 1080p. To measure computational performance, each model processed videos at the specified resolutions while we recorded the FPS, CPU usage, and GPU usage. This allowed us to analyze how each optimization affected the model's ability to handle higher resolutions, which is critical for real-time applications. The experiments were conducted under consistent hardware conditions to ensure a fair comparison among the models. Table V presents the computational performance of each model across the different resolutions. From the results, we observe the following:

- **FPS Improvement:** The PPFP16 and Multithreading models significantly increase FPS, especially at lower resolutions. For instance, at 320p, the Multithreading model achieves an FPS of 66.55, more than double that of the unoptimized SWCNN [8]. However, as resolution increases, the FPS advantage diminishes.
- **Resource Utilization:** The Multithreading model maintains low CPU and GPU utilization despite the higher FPS, indicating efficient resource management.
- **Trade-Offs:** Despite the efficiency gains, the earlier visual assessments revealed that the PPFP16 and Multithreading models may compromise image quality.

D. Ablation Studies

To further understand the impact of each optimization, we conducted ablation studies focusing on the effects of multithreading and varying watermark sizes on model performance.

1) Impact of Multithreading: We isolated the effect of multithreading by comparing the performance and image quality of the SWCNN with PPFP16 against the SWCNN with multithreading optimization. Both models share the same preprocessing optimizations, but the latter includes multithreading to enhance processing speed. The multithreading model showed a substantial increase in FPS, improving processing speed by approximately 36% at 720p resolution compared to the PPFP16 model. However, this gain in speed was accompanied by a noticeable drop in PSNR and SSIM values, indicating a reduction in image quality. Visual assessments revealed residual watermark artifacts and image inconsistencies. These findings suggest that while multithreading enhances computational efficiency, it may introduce synchronization issues or data handling challenges that negatively affect image quality.

2) Effect of Watermark Size: We evaluated the models' performance with varying watermark sizes to assess robustness under different conditions. Using the same experimental setup as in Section V-B1, we applied watermarks of different sizes to the test videos and measured the PSNR and SSIM values after processing. As watermark size increased, both models experienced a decrease in performance, with larger watermarks posing a greater challenge for removal. The SWCNN with FP16 Computation consistently outperformed the other models across all watermark sizes, demonstrating greater robustness.

TABLE V: Computational Performance at Different Resolutions

Model	320p (FPS / CPU% / GPU%)	480p (FPS / CPU% / GPU%)	720p (FPS / CPU% / GPU%)	1080p (FPS / CPU% / GPU%)
SWCNN [8]	27.76 / 1.4% / 2.9%	12.89 / 42.1% / 2.3%	7.21 / 3.1% / 2.1%	3.37 / 3.0% / 2.1%
SWCNN+FP16	32.53 / 1.8% / 3.5%	16.34 / 2.4% / 2.7%	10.11 / 3.1% / 2.5%	5.39 / 3.3% / 2.4%
SWCNN+PPFP16	64.64 / 0.6% / 2.7%	31.52 / 1.8% / 1.9%	13.05 / 2.6% / 1.6%	6.93 / 3.3% / 1.5%
SWCNN+Multith	66.55 / 0.6% / 2.7%	34.98 / 1.7% / 1.9%	17.82 / 2.5% / 1.6%	11.74 / 3.3% / 1.5%

E. Real-Time Watermark Removal on Local and Online Streams

Building on the experiments described above, we integrated FP16-based optimizations to handle real-time watermark removal across both locally streamed and online video sources. This subsection presents qualitative outcomes and performance insights, focusing first on the local live streaming environment and then on Twitch-based online streams. Finally, we compare the two setups to highlight the key factors influencing performance.

1) Local Live Streaming Results: Under controlled local streaming conditions, using FP16 optimization and a watermark capacity of ($\alpha = 0.5$), the pipeline attains consistent, high-definition playback with minimal delay. Table VI summarizes key performance indicators. In this scenario, a frame rate of approximately 26.31 FPS was achieved with CPU usage around 44.20% and GPU usage at 3.51%. The PSNR of 38.56 indicates robust watermark removal quality. Fig. 9 presents a visual comparison of frames before and after watermark removal, demonstrating that even under ideal conditions, the pipeline delivers artifact-free, real-time streams.

2) Online Streaming Platform (Twitch) Results: In the case of Twitch-based streaming, the pipeline faces challenges such as unstable network conditions, fluctuating bitrates, and dynamic GOP structures. Even so, the system maintains real-time watermark removal at approximately 10.89 FPS, albeit at lower resolutions to mitigate stuttering. CPU and GPU usage remain low at around 2.10% and 2.68%, respectively, illustrating that resource demands are modest. Notably, the PSNR of 40.9 suggests that watermark removal quality remains high, despite variable stream quality.

Beyond standard watermark removal, we further tested the pipeline's relevance by overlaying simulated illicit content—such as unauthorized online gambling advertisements onto the cleaned Twitch feed. This scenario, illustrated in Fig. 10, demonstrates how the pipeline could be exploited in real-world illicit re-streaming or content manipulation scenarios. Despite reduced playback smoothness at HD or 1080p levels, the model adapts to network fluctuations, preserving a consistently high quality of watermark removal while facilitating a controlled introduction of additional content.

3) Comparison and Analysis: Comparing local (VLC) and online (Twitch) results highlights the influence of external factors. While local streams achieve a stable 26.31 FPS under FP16 optimization with high-resolution playback and low latency, Twitch streams settle at around 10.89 FPS to remain fluid. The reduced frame rate and occasional buffering in the online setting highlight the need for adaptive strategies to handle volatile network conditions. Nevertheless, the sustained watermark removal quality—evidenced by high PSNR values in both cases—confirms the pipeline's effectiveness.

From these experiments, it is evident that FP16 optimization provides a balanced approach to real-time watermark removal across diverse streaming conditions. The local scenario demonstrates the system's full potential, offering high-quality, low-latency output. The Twitch scenario, though more demanding, still achieves reliable watermark removal and even allows for realistic, albeit nefarious, content manipulation applications. Future refinements, such as dynamic resolution scaling, advanced buffering strategies, or integration with more robust content adaptation techniques, could further improve performance and applicability in complex, real-world streaming environments.

TABLE VI: Performance Summary with FP16 Optimization and ($\alpha = 0.5$) Watermark Capacity

Scenario	FPS	CPU Usage	GPU Usage	PSNR
Local (VLC)	26.31	44.20%	3.51%	38.56
Online (Twitch)	10.89	2.10%	2.68%	40.9

Fig. 9: Local streaming scenario: Example frames before and after watermark removal. High-resolution and minimal latency conditions yield artifact-free output.

F. Limitations and Trade-Offs

Our experiments reveal a trade-off between computational efficiency and image quality in the optimization of the SWCNN models.

- **FP16 Computation:** This optimization offers a balanced improvement in processing speed and image quality. It enhances computational efficiency without introducing significant artifacts, making it suitable for most applications where both speed and quality are important.
- **PPFP16 and Multithreading:** These optimizations significantly boost processing speed but may compromise the effectiveness of watermark removal, resulting in degraded image quality and residual artifacts.

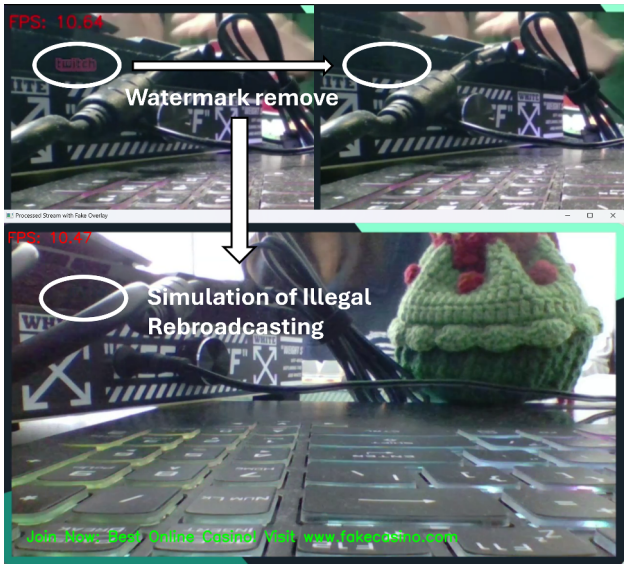

Fig. 10: Twitch streaming scenario: Example frames before and after watermark removal, including the overlay of simulated illicit content (online gambling ads). While network conditions necessitate lower resolution and may introduce slight latency, watermark removal quality remains robust.

The benefits of multithreading are contingent on the availability of sufficient CPU resources, and introducing advanced optimizations increases code complexity. Further research is needed to explore adaptive strategies that balance speed and quality based on resource availability and application requirements. Future research could focus on optimizing multithreading implementations to minimize their impact on image quality. Additionally, exploring advanced synchronization techniques or alternative parallel processing frameworks may help mitigate the negative effects on image quality while retaining computational efficiency gains.

VI. Discussion

In real-world live streaming environments, several challenges differentiate practical deployment from controlled offline tests. Network latency, adaptive bitrate changes, and intermittent packet loss can introduce fluctuations that directly affect frame acquisition rates, synchronization, and overall visual quality. Although our pipeline's integration of half-precision (FP16) inference, preprocessing enhancements, and multithreading significantly improves inference speed and throughput, these optimizations must contend with the inherent variability of live network conditions.

The benefits of FP16 optimization are pronounced in scenarios where GPU resources are limited, as it reduces both computation time and memory usage. Preprocessing steps such as dynamic resizing and normalization prove invaluable when streams experience changing resolutions and codecs. Meanwhile, multithreading effectively mitigates frame capture or display bottlenecks, ensuring a more responsive user experience. However, further refinements are needed to fully exploit these optimizations. For instance, implementing adaptive resolution scaling or advanced buffering strategies could stabilize frame rates under shifting network conditions.

This research highlights that real-time watermark removal can be feasibly integrated into diverse streaming sources, including locally served content and third-party platforms like Twitch. By demonstrating robust watermark removal even in the presence of bitrate fluctuations, dynamic GOP structures, and mild stuttering at high resolutions, we reaffirm the pipeline's adaptability. More importantly, this study underscores the significance of advancing content protection mechanisms. As watermark removal attacks improve, content owners must bolster their defense strategies, ensuring that evolving watermarking techniques can withstand increasingly sophisticated attacks.

VII. Conclusion

A. Summary

In this paper, we presented an optimized approach to real-time video watermark removal using an enhanced SWCNN model. We validated our methods not only on offline videos but also in real-time scenarios, including locally served streams and live feeds from external platforms such as Twitch. The integration of half-precision (FP16) computation, preprocessing enhancements, and multithreading techniques helped achieve a balance between efficiency and quality.

Through extensive experiments, our FP16-optimized SWCNN delivered improved inference speed while maintaining high watermark removal fidelity. It achieved enhanced PSNR and SSIM metrics compared to the unoptimized model and effectively preserved image details, as confirmed by visual assessments. Although the PPFP16 and multithreading optimizations significantly increased FPS, these gains sometimes came at the expense of image quality. These findings highlight the importance of tailoring optimization strategies to specific application requirements, especially when balancing speed and fidelity is critical.

Our comprehensive analysis, performed both under controlled local conditions and on actual Twitch streams, demonstrates that the proposed framework can adapt to a range of operating environments. By showing that effective real-time watermark removal is possible even with dynamic network conditions and complex streaming protocols, this work provides valuable insights into defending copyrighted material against unauthorized redistribution.

B. Future Work

Building upon the findings of this study, future research can pursue several directions to enhance real-time video watermark removal attacks and improve defense mechanisms against illicit broadcasting:

- **Adaptive Optimization Techniques:** One may investigate dynamic strategies that adjust between speed and quality in response to fluctuating network conditions and hardware constraints.

- **Advanced Multithreading and Buffering:** We can also further explore more refined parallel processing frameworks and intelligent buffering schemes to stabilize frame rates, especially under challenging online conditions.
- **Broadening Watermark Variability:** On ecan also extend the approach to handle a wider array of watermark types, patterns, and dynamic behaviors encountered in diverse real-world scenarios.
- **Scaling Up Resolutions:** Optimizing the model for ultra-high-definition (UHD) and beyond and maintaining real-time performance and robust watermark removal is also a future research direction. This enables experimenting with emerging neural network architectures, such as attention-based models, to further improve efficiency and adaptability.
- **Strengthening Copyright Protection:** Finally, the community need to employ insights gleaned from this attacker's perspective to guide the development of more secure watermarking techniques. Ensuring resilience against advanced removal attacks will reinforce copyright protection, forensic evidence collection, and reliable traceability across complex streaming ecosystems.

In summary, this research not only enhances real-time watermark removal attacks but also informs the design of more resilient watermarking systems. By bridging the gap between offline experiments and challenging real-world streaming environments, we contribute to a stronger, more secure framework for protecting streaming content and combating illicit redistribution. As live streaming continues to evolve, further innovation in adaptive optimization, network-aware strategies, and scalable architectures will be required to ensure robust and reliable content protection.

ACKNOWLEDGMENT

The authors would like to thank all contributors to open-source datasets and libraries used in this research.

REFERENCES

[1] S. Byun, H. Son, and S. Lee, "Fast and Robust Watermarking Method Based on DCT Specific Location," *IEEE Access*, vol. 7, pp. 100706-100718, 2019, doi: 10.1109/ACCESS.2019.2931039.

[2] C. Chen, W. Yin, Z. Huang, and S. Shi, "AGiLE: Enhancing adaptive GOP in live video streaming," *Proceedings of the 15th ACM Multimedia Systems Conference*, 2024, pp. 34–44, doi: 10.1145/3625468.3647609.

[3] "Manage your channel branding," YouTube Help, accessed 2024. [Online]. Available: https://support.google.com/youtube/answer/10456525?hl=en&co=GENIE.Platform%3DDesktop#zippy=%2Cadd-your-video-watermark

[4] Y. Zhou, C. Wang, and X. Zhou, "An Intra-Drift-Free Robust Watermarking Algorithm in High Efficiency Video Coding Compressed Domain," *IEEE Access*, vol. 7, pp. 132991-133007, 2019, doi: 10.1109/ACCESS.2019.2940366.

[5] "Deep-Live-Cam," GitHub repository, accessed 2024. [Online]. Available: https://github.com/hacksider/Deep-Live-Cam.

[6] R. Chesney and D. Citron, "Deepfakes and the new disinformation war: The coming age of post-truth geopolitics," *Foreign Affairs*, vol. 98, no. 1, pp. 147–155, 2019.

[7] Y. Liu, Z. Zhu, and X. Bai, "WDNet: Watermark de-rendering network for visible watermark removal," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2021, pp. 3685-3693.

[8] C. Tian, M. Zheng, T. Jiao, W. Zuo, Y. Zhang, and C.-W. Lin, "A Self-Supervised CNN for Image Watermark Removal," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 34, no. 8, pp. 7566-7576, Aug. 2024, doi: 10.1109/TCSVT.2024.3375831.

[9] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. S. Huang, "Generative image inpainting with contextual attention," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5505–5514.

[10] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *arXiv preprint arXiv:1803.01271*, 2018.

[11] H. Zhang, I. Goodfellow, D. Metaxas, and A. Odena, "Self-attention generative adversarial networks," in *International Conference on Machine Learning*, 2019, pp. 7354–7363.

[12] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, no. 1, p. 60, 2019.

[13] C. Wang, P. Tian, Z. Wei, Q. Li, Z. Xia, and B. Ma, "CAWNet: A Channel Attention Watermarking Attack Network Based on CWABlock," in *Pattern Recognition and Computer Vision. PRCV 2023. Lecture Notes in Computer Science*, vol. 14433. Springer, Singapore, 2024, pp. 34-44, doi: 10.1007/978-981-99-8546-3_4.

[14] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 7132–7141.

[15] S. Woo, J. Park, J. Y. Lee, and I. S. Kweon, "CBAM: Convolutional block attention module," in *Proceedings of the European Conference on Computer Vision (ECCV)*, 2018, pp. 3–19.

[16] C. Tian, Y. Xu, L. Fei, and W. Zuo, "Image denoising using deep CNN with batch renormalization," *Neural Networks*, vol. 121, pp. 461–473, 2020.

[17] P. Micikevicius et al., "Mixed precision training," *International Conference on Learning Representations*, 2018.

[18] G. Henry, P. Tang, and A. Heinecke, "Leveraging the bfloat16 Artificial Intelligence Datatype For Higher-Precision Computations," in *2019 IEEE 26th Symposium on Computer Arithmetic (ARITH)*, 2019, pp. 69-76, doi: 10.1109/ARITH.2019.00019.

[19] NVIDIA Corporation, "NVIDIA Tensor Cores," 2020. [Online]. Available: https://www.nvidia.com/en-us/data-center/tensor-cores/.

[20] D. Kalamkar et al., "A study of BFLOAT16 for deep learning training," *arXiv preprint arXiv:1905.12322*, 2019.

[21] T. Ben-Nun and T. Hoefler, "Demystifying parallel and distributed deep learning: An in-depth concurrency analysis," *ACM Computing Surveys*, vol. 52, no. 4, pp. 1–43, 2019.

[22] J. Chen, K. Li, Q. Deng, K. Li, and P. S. Yu, "Distributed Deep Learning Model for Intelligent Video Surveillance Systems with Edge Computing," *IEEE Transactions on Industrial Informatics*, 2019, doi: 10.1109/TII.2019.2909473.

[23] B. Jacob et al., "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.

[24] X. Zhang, X. Zhou, M. Lin, and J. Sun, "ShuffleNet: An extremely efficient convolutional neural network for mobile devices," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6848–6856.

[25] Q. Li et al., "Weighted Res-UNet for High-Quality Retina Vessel Segmentation," in *2019 IEEE International Conference on Computer Vision Workshops (ICCVW)*.

[26] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, "Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising," *IEEE Transactions on Image Processing*, vol. 26, no. 7, pp. 3142-3155, July 2017, doi: 10.1109/TIP.2017.2662206.

[27] S. Ioffe, "Batch Renormalization: Towards Reducing Minibatch Dependence in Batch-Normalized Models," in Advances in Neural Information Processing Systems (NeurIPS), 2017, pp. 1945–1953.

[28] P. Micikevicius, S. Narang, J. Alben, G. Diamos, E. Elsen, D. Garcia, B. Ginsburg, M. Houston, O. Kuchaiev, G. Venkatesh, and H. Wu, "Mixed Precision Training," in *arXiv preprint*, 2018. Available: https://arxiv.org/abs/1710.03740.

[29] K. Khandekar, "Popular Brand Logos Image Dataset," Kaggle, 2020. Retrieved from https://www.kaggle.com/datasets/kkhandekar/popular-brand-logos-image-dataset..

[30] ShareGPTVideo. (2022). Test Raw Video Data. Hugging Face. Retrieved from https://huggingface.co/datasets/ShareGPTVideo/test_raw_video_data
[31] K. Zhang, W. Zuo, and L. Zhang, "Ffdnet: Toward a fast and flexible solution for cnn-based image denoising," *IEEE Transactions on Image Processing*, vol. 27, no. 9, pp. 4608–4622, 2018.
[32] S. Deng, M. Wei, J. Wang, L. Liang, H. Xie, and M. Wang, "Drdnet: Detail-recovery image deraining via context aggregation networks," *arXiv preprint arXiv*:1908.10267, 2019.
[33] X. Wang, Z. Li, H. Shan, Z. Tian, Y. Ren, and W. Zhou, "Fastderainnet: A deep learning algorithm for single image deraining," *IEEE Access*, vol. 8, pp. 127 622–127 630, 2020.
[34] C. Sun, H. Lai, L. Wang, and Z. Jia, "Efficient attention fusion network in wavelet domain for demoireing," *IEEE Access*, vol. 9, pp. 53 392–53 400, 2021.
[35] Y. Leng, C. Fang, J. Chen, Y. Fang, S. Li, and G. Li, "Bridging knowledge gap between image inpainting and large-area visible watermark removal," *arXiv preprint arXiv:2504.04687*, 2025. [Online]. Available: https://arxiv.org/abs/2504.04687
[36] M. Peter, S. Seifi, and S. Esedoglu, "Deep spatial and tonal data optimisation for homogeneous diffusion inpainting," *arXiv preprint arXiv:2208.14371*,
[37] R. Xu, X. Yang, X. Zhou, and S. Wu, "Texture memory-augmented deep patch-based image inpainting," *arXiv preprint arXiv:2009.13240*, 2020. [Online]. Available: https://arxiv.org/abs/2009.13240
[38] Z. Yang and R. Ruhaiyem, "Review of deep learning-based image inpainting techniques," *ResearchGate preprint*, 2024. [Online]. Available: https://www.researchgate.net/publication/384536208_Review_of_Deep_Learning-Based_Image_Inpainting_Techniques
[39] Github link. (2025). https://github.com/N3K0521/Real-Time-Video-Watermark-Removal-using-Optimized-SWCNN