Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Supplementary materials for

Hui SHI, Guibin WANG, Yanni LI, Rujia QI, 2025. Full-defense framework: multi-level deepfake detection and source tracing. *Front Inform Technol Electron Eng*, 26(9):1649-1661.
https://doi.org/10.1631/FITEE.2401012

1 Proactive defense module

The SepMark module includes the encoder En, the noise layer NL, and the separable decoders Tr and De. The structure is shown in Fig. S1. An encoder En embeds a watermark into the image. The robust decoder Tr can resist various attacks, while the semi-robust decoder De is sensitive to malicious distortions and cannot extract a complete watermark. By comparing the extracted watermarks based on the robust decoder Tr and the semi-robust decoder De, the authenticity of the image can be determined.

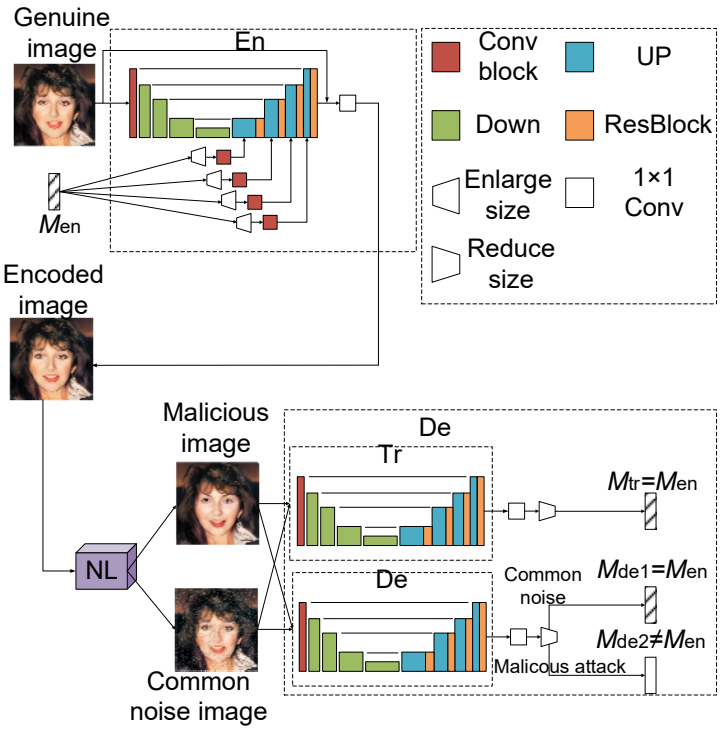

Fig. S1 Separable watermark proactive defense module

1.1 Encoder

The encoder uses a U-net-like architecture, including ConvBlock, ResBlock, Down module, and UP module.

The ConvBlock is constructed from multiple linearly stacked ConvINRelu sub-modules. The operation of ConvINRelu and ConvBlock is demonstrated in Eqs. (S1) and (S2):

$$\mathrm{ConvINRelu}(\boldsymbol{f}) = \mathrm{Relu}\left(\mathrm{IN}\left(\mathrm{Conv}(\boldsymbol{f})\right)\right), \tag{S1}$$

$$\mathrm{ConvBlock}(\boldsymbol{f}) = \prod_{i=1}^{n} \mathrm{ConvINRelu}_i(\boldsymbol{f}), \tag{S2}$$

where Relu denotes the activation function, IN represents the instance normalization layer, Conv signifies the convolutional layer, and $\boldsymbol{f}$ denotes the input feature map.

The ResBlock enhances the standard residual blocks by using the InstanceNorm2d instance normalization function instead of the BatchNorm2d batch normalization function.

The Down module consists of two ConvBlock, as shown in Eq. (S3). The UP module comprises two steps, as shown in Eq. (S4). Initially, the feature map $\boldsymbol{f}$ undergoes nearest neighbor interpolation to enlarge it fourfold. Subsequently, the enlarged feature map $\boldsymbol{f}$ is fed into ConvBlock:

$$\mathrm{Down}(\boldsymbol{f}) = \mathrm{ConvBlock}_2, \left(\mathrm{ConvBlock}_1(\boldsymbol{f})\right), \tag{S3}$$

$$\mathrm{UP}(\boldsymbol{f}) = \mathrm{ConvBlock}\left(\mathrm{interpolate}(\boldsymbol{f})\right), \tag{S4}$$

where interpolate() signifies nearest neighbor interpolation, and $\boldsymbol{f}$ represents the feature map.

The encoding process involves downsampling, upsampling, and concatenation. The downsampling process is depicted as follows:

$$\boldsymbol{f}_0 = \mathrm{ConvBlock}(I_{\mathrm{CO}}), \tag{S5}$$

$$\boldsymbol{f}_i = \mathrm{Down}(\boldsymbol{f}_{i-1}), i \in [1,3], \tag{S6}$$

$$\boldsymbol{u}_4 = \mathrm{Down}(\boldsymbol{f}_3), \tag{S7}$$

where I_{co} denotes the cover image, and $\boldsymbol{f}_0$, $\boldsymbol{f}_i$, and $\boldsymbol{u}_4$ represent feature vectors with different sizes. The cover image I_{co} is fed into ConvBlock for feature extraction, yielding $\boldsymbol{f}_0$. Subsequently, four Down modules are followed, where the outcomes of the first three Down modules are termed $\boldsymbol{f}_i$, and the output of the last Down module is termed $\boldsymbol{u}_4$.

Subsequently, $\boldsymbol{u}_i$ is fed into the UP module for upsampling. After processing by the UP module, the input message sequence M is enlarged to match the sizes of $\boldsymbol{f}_{i\text{-}1}$ and $\boldsymbol{u}_{i\text{-}1}$ through linear transformation, interpolation, and ConvBlock. Then, $\boldsymbol{f}_{i\text{-}1}$ and $\boldsymbol{u}_{i\text{-}1}$ are concatenated and fed into ResBlock, resulting in the transformed $\boldsymbol{u}_{i\text{-}1}$, as described below:

$$\boldsymbol{u}_{i-1} = \mathrm{UP}(\boldsymbol{u}_i), i \in \{4,3,2,1\}, \tag{S8}$$

$$\boldsymbol{u}_{i-1} = \mathrm{ResBlock}(\mathrm{Cat}(\boldsymbol{f}_{i-1}, \boldsymbol{u}_{i-1}, \boldsymbol{M}')), \tag{S9}$$

where Cat denotes concatenation and $\boldsymbol{M}'$ represents the enlarged size of the message sequence.

Finally, the cover image $\boldsymbol{I}_{\mathrm{co}}$ and $\boldsymbol{u}_0$ are concatenated, and the number of channels of the concatenated result is adjusted to 3 by a 1×1 convolution, completing the watermark embedding to produce the initial encoded image $\boldsymbol{I}'$ The initial encoded image $\boldsymbol{I}'$ is subtracted from the cover image $\boldsymbol{I}_{\mathrm{co}}$, multiplied by the embedding strength coefficient δ, and then added to the cover image $\boldsymbol{I}_{\mathrm{co}}$ to obtain the final encoded image $\boldsymbol{I}_{\mathrm{en}}$, as shown in Eqs. (26) and (27):

$$\boldsymbol{I}' = \mathrm{Conv}(\mathrm{Cat}(\boldsymbol{I}_{\mathrm{co}}, \boldsymbol{u}_0)), \tag{S10}$$

$$\boldsymbol{I}_{\mathrm{en}} = \boldsymbol{I}_{\mathrm{co}} + \delta(\boldsymbol{I}' - \boldsymbol{I}_{\mathrm{co}}). \tag{S11}$$

1.2 Noise layer

The noise layer introduces various distortions to the encoded image, simulating the effects of noise and malicious attacks during image transmission. Multiple types of distortions are incorporated and jointly trained in the noise layer to ensure robustness against different attacks. Given the noised image $\boldsymbol{I}_{\text{no}}$, the real distortion gap can be directly obtained as gap=$\boldsymbol{I}_{\text{no}}$−$\boldsymbol{I}_{\text{en}}$ in a detached forward propagation. Thus, both standard forward and backward propagation can be realized through differentiable $\boldsymbol{I}_{\text{no}}$=gap+$\boldsymbol{I}_{\text{en}}$. Specifically, random sampling from both common and malicious distortions, only common distortions, and only malicious distortions results in three batches of $\boldsymbol{I}_{\text{no}}$.

1.3 Separable decoders

The decoders of the SepMark comprise a robust decoder Tr and a semi-robust decoder De, which share identical network structures but differ in their weights, as illustrated in the lower part of Fig. 3. Two decoders perform watermark extraction from noisy images twice, with three main stages: downsampling, upsampling, and concatenation.

Initially, the encoded image $\boldsymbol{I}_{\text{en}}$ is fed into ConvBlock to obtain the feature map $\boldsymbol{f}_0$. Subsequently, it sequentially passes through four Down modules. The process is illustrated as follows:

$$\boldsymbol{f}_0 = \text{ConvBlock}(\boldsymbol{I}_{\text{en}}), \tag{S12}$$

$$\boldsymbol{f}_i = \text{Down}(\boldsymbol{f}_{i-1}), i \in [1,3], \tag{S13}$$

$$\boldsymbol{u}_4 = \text{Down}(\boldsymbol{f}_3). \tag{S14}$$

Next, $\boldsymbol{u}_i$ is input to the UP module for upsampling, resulting in the feature map $\boldsymbol{u}_{i-1}$. Then, $\boldsymbol{f}_{i-1}$ and $\boldsymbol{u}_{i-1}$ are concatenated and processed through ResBlock to produce the transformed $\boldsymbol{u}_{i-1}$, as follows:

$$\boldsymbol{u}_{i-1} = \text{UP}(\boldsymbol{u}_i), i \in \{4,3,2,1\}, \tag{S15}$$

$$\boldsymbol{u}_{i-1} = \text{ResBlock}(\text{Cat}(\boldsymbol{f}_{i-1}, \boldsymbol{u}_{i-1})). \tag{S16}$$

Finally, the channel of $\boldsymbol{u}_0$ is adjusted to 1 using a 1×1 convolution, and $\boldsymbol{u}_0$ is resized to [len$_M$×len$_M$] using nearest neighbor interpolation to obtain $\boldsymbol{u}_0'$, which is then fed through linear transformation to produce a one-dimensional matrix, yielding the decoded message sequence $\boldsymbol{M}''$.

1.4 Adversarial discriminator

Adversarial training is used to enhance the visual quality of the encoded image. The discriminator is trained alternately with the encoder and decoder using a PatchGAN network, which focuses on local image regions to capture fine-grained features. By optimizing the loss, the PatchGAN network distinguishes between original images and stego-images, aiding the encoder in improving the visual quality of the encoded image. Initially, the image is segmented into multiple local regions (patches). Then, each patch is processed through convolution layers and activation functions to extract local features. Finally, a 1×1 convolution is used to output the final predicted image.

During training, we process images in mini-batches of size B=16, where the discriminator is updated using the loss term L_{Ad1} to learn the relative authenticity between the cover and encoded image pairs. This relative comparison approach helps stabilize the training process and effectively prevents mode collapse issues commonly encountered in standard GANs. The encoder network is simultaneously optimized through L_{Ad2}, which encourages the generation of encoded images that are perceptually indistinguishable from their cover counterparts. The average operation across batch samples ($\frac{1}{B}\sum_{i=1}^{B} I_i$) provides smoothed gradients and enhanced training stability. To ensure optimal performance, we implement gradient penalty for maintaining the Lipschitz continuity of the discriminator, and the learning rates are adaptively adjusted based on the relative magnitudes of

different loss components. The adversarial loss terms (L_{Ad1} and L_{Ad2}) work in synergy with other objective functions (L_{En}, L_{Tr}, L_{De1}, L_{De2}, L_P), collectively optimized through carefully balanced weights (λ_1, λ_2, λ_3, λ_4, λ_5, λ_6) in the total loss L_{Total}. This comprehensive adversarial training strategy significantly contributes to both the imperceptibility of the embedded watermarks and the robustness of our framework against various potential attacks, while maintaining high visual quality of the encoded images.

2 Algorithm

2.1 Overall procedure of the method

The overall procedure of the method is given in Algorithm S1.

Algorithm S1 Overall procedure of the method

Input: datasets D; encoder En; decoder Tr, De; passive detection module P; detection vector $\boldsymbol{V}$
for $\boldsymbol{I}_{\mathrm{co}}$ in D
  $\boldsymbol{I}_{\mathrm{en}}$=En($\boldsymbol{I}_{\mathrm{co}}$, $\boldsymbol{M}_{\mathrm{en}}$)
  $\boldsymbol{I}_{\mathrm{no}}$=Common_noise($\boldsymbol{I}_{\mathrm{en}}$)
  $\boldsymbol{I}_{\mathrm{df}}$, $\boldsymbol{I}_{\mathrm{wr}}$=Deep_fake&Watermark_removal($\boldsymbol{I}_{\mathrm{en}}$)
  $\boldsymbol{M}_{\mathrm{tr}}$=Tr($\boldsymbol{I}_{\mathrm{no}}$, $\boldsymbol{I}_{\mathrm{df}}$, $\boldsymbol{I}_{\mathrm{wr}}$)
  M_{de}, 0, 0=De(I_{no}, I_{df}, I_{wr})
  $\boldsymbol{V}$[False, True]=P($\boldsymbol{I}_{\mathrm{df}}$, $\boldsymbol{I}_{\mathrm{wr}}$)
end for
Output: $\boldsymbol{V}$, $\boldsymbol{M}_{\mathrm{tr}}$, $\boldsymbol{M}_{\mathrm{de}}$

2.2 Training code

Training code is given in Algorithm S2.

Algorithm S2 Training procedure of the method

Input: datasets D; adversarial discriminator Ad; encoder En; decoder Tr, De; passive detection module P; number of training epochs T
// training stage
while $t \leq T$ **do**
  for $\boldsymbol{I}_{\mathrm{co}}$, label **in** D
  // Train Ad
    $L_2\left(\mathrm{Ad}\left(\boldsymbol{\omega}_{\mathrm{ad}}, \boldsymbol{I}_{\mathrm{co}}\right) - \overline{\mathrm{Ad}}\left(\boldsymbol{\omega}_{\mathrm{ad}}, \boldsymbol{I}_{\mathrm{en}}\right), 1\right) + L_2\left(\mathrm{Ad}\left(\boldsymbol{\omega}_{\mathrm{ad}}, \boldsymbol{I}_{\mathrm{en}}\right) - \overline{\mathrm{Ad}}\left(\boldsymbol{\omega}_{\mathrm{ad}}, \boldsymbol{I}_{\mathrm{co}}\right), -1\right)$
  // Fixing Ad, train En Tr, De, and P
    $\lambda_1 L_{\mathrm{Ad2}} + \lambda_2 L_{\mathrm{En}} + \lambda_3 L_{\mathrm{Tr}} + \lambda_4 L_{\mathrm{De1}} + \lambda_5 L_{\mathrm{De2}} + \lambda_6 L_P$
  end for
end while
Output: Ad, En, De, and P

3 Visual quality test

Fig. S2 presents the visual quality under various attacks, including Identity, Resize, Gaussian Blur, Median Blur, Brightness, Contrast, Saturation, Hue, Dropout, Salt & Pepper Noise, Gaussian Noise, Deepfake, and FaceSwap, while a watermark removal attack is given in Fig. S3. From top to bottom, Figs. S2 and S3 show the cover image $\boldsymbol{I}_{\mathrm{co}}$, the encoded image $\boldsymbol{I}_{\mathrm{en}}$, the attacked image $\boldsymbol{I}_{\mathrm{no}}$, the residual signal of $N(|\boldsymbol{I}_{\mathrm{co}}-\boldsymbol{I}_{\mathrm{en}}|)$, and $N(|\boldsymbol{I}_{\mathrm{no}}-\boldsymbol{I}_{\mathrm{en}}|)$, where $N(\boldsymbol{I})=(\boldsymbol{I}-\min(\boldsymbol{I}))/(\max(\boldsymbol{I})-\min(\boldsymbol{I}))$. Each column represents a type of attack, with an image size of 256×256. From Figs. S2 and S3, there are no noticeable artifacts between the encoded images and cover images.

Our algorithm successfully extracts the watermark under common attacks (Identity, JPEG, Resize, Gaussian Blur, Median Blur, Brightness, Contrast, Saturation, Hue, Dropout, Salt & Pepper Noise, Gaussian Noise), allowing source tracing and copyright verification. However, under deepfake attacks (Deepfake, FaceSwap) and watermark removal attack (SWCNN), the watermark cannot be extracted, which activates the proposed deepfake detection module.

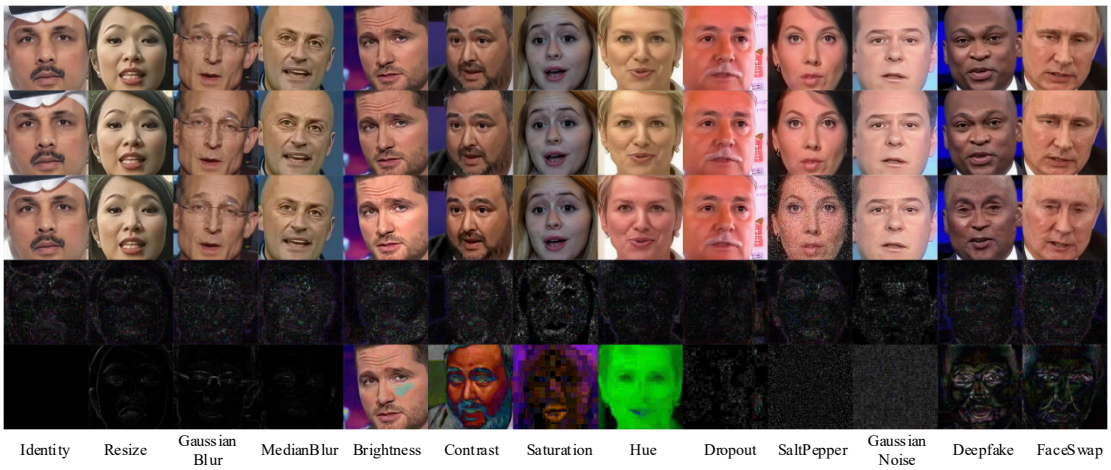

Fig. S2 Visual quality under common and malicious deepfake attacks

Fig. S3 Visual quality under watermark removal attacks