R+R: Demystifying ML-Assisted Side-Channel Analysis Framework: A Case of Image Reconstruction

Zhiyuan Zhang*, Zhenzhi Lai*, Udaya Parampalli
The University of Melbourne
Melbourne, Australia
{*zhiyuanz5, zhenzhil*}*@student.unimelb.edu.au, udaya@unimelb.edu.au*

***Abstract*—Machine-learning-assisted side-channel analysis (ML-assisted SCA) automates the procedure of analyzing side-channel activities to reconstruct secrets. Although ML-assisted SCA does produce promising results, it is hard to determine whether its machine-learning model tends to reconstruct secrets or generate new instances. In this paper, we revisit the first general ML-assisted SCA framework for media software (Yuan et al. USENIX Security 2022), which we refer to as the Manifold-SCA framework, with a case study of reconstructing images from cache activities. We show that Manifold-SCA tends to generate images more than reconstruct them. Inspired by the autoencoder implemented in the Manifold-SCA framework, we theoretically and experimentally show that an autoencoder is sufficient to reconstruct images from cache activities. Through three ablation studies, we show that an autoencoder outperforms the Manifold-SCA framework under all scenarios. In the end, we apply an autoencoder to analyze practical cache activities collected by a profiling-based Prime+Probe attack, and show that an autoencoder can reconstruct partial pixel-related activities, but these activities are insufficient to reconstruct images due to the information loss in the activities.**

1. Introduction

Side-channel attacks collect activities of victim software on shared hardware resources to infer secrets [1, 2, 3, 4, 5, 6]. The procedure of analyzing side-channel activities is called side-channel analysis (SCA). SCA is often done manually to locate leakages and build algorithms to reconstruct secrets. In recent years, due to the rapid development of machine-learning techniques, machine-learning-assisted SCA (ML-assisted SCA) is proposed to automate the analysis procedure [7, 8, 9, 10, 11, 12, 13, 14]. Such as facilitating website fingerprinting attacks [7, 8, 9] or power analysis attacks [10, 11, 15, 16, 17]. Although ML-assisted SCA describes a bright future for side-channel analysis, many existing approaches focus on analyzing specific side-channel activities to infer particular secrets. Namely, there is a lack of a general framework to analyze various side-channel activities to recover various types of secrets.

Such a situation has been changed since 2022, when Yuan et al. [12] proposed the first general ML-SCA framework that can analyze multiple micro-architecture activities to reconstruct various types of media inputs, such as images, texts, and audio. Unlike traditional SCA on media software [18, 19], where an attacker reconstructs inputs pixel-by-pixel, Yuan et al. [12] hypothesize that the perception-level (e.g., hairstyle) connection between media inputs and activities resides in a joint manifold space. The proposed framework leverages a combination of an autoencoder [20, 21] and a discriminator network [22, 23, 24, 25] to learn this space and reconstruct media inputs from side-channel activities. Notably, despite the novel design of the framework, another contribution is its ability to automatically analyze side-channel activities without knowledge of the victim software. That is, the victim software is treated as a black-box during the automated analysis.

In this paper, we revisit the first ML-assisted SCA framework for media software [12], which we refer to as the Manifold-SCA Framework, with a case study of reconstructing images from cache activities. Through our reproducibility and replicability evaluation, we find that the reported performance of the framework was overly optimistic due to a mistake in interpreting the evaluation metric. Furthermore, we notice that the framework, a deep-learning model, is trained in a manner similar to generative adversarial networks (GANs) [26], which are known for generating new instances from its training set. It is problematic in side-channel analysis, because the goal of SCA is to *reconstruct* secrets, but not *generate* new instances. However, the boundary between reconstruction and generation is vague. We only have confidence to say an output of a model is reconstructed when it is identical to its reference data (original data). When they are not identical, we cannot conclude if the different parts are incorrectly reconstructed or newly generated.

To determine the extent of reconstruction and generation of an image reconstructed by ML-assisted SCA, we leverage the similarity among reconstructed images, their reference images, and other images. We note that a reconstructed image should be more similar (or even identical) to its reference image than other images. Therefore, we measure the similarity between a reconstructed image and its reference

* *Equal contribution. Contact Zhiyuan Zhang for coorespondence.*

image and compare the result with that between it and other images. We name this evaluation metric as *distinguishability evaluation*. A well-reconstructed image should have higher similarity with its reference image than other images, while a generated image has no specific relationship with a single image.

Our evaluation shows that the Manifold-SCA framework tends to generate images more than reconstruct them. Being inspired by the autoencoder in the Manifold-SCA framework, we hypothesize that an autoencoder alone is sufficient to reconstruct images, while a discriminator network misguides it to generate images. We compare the image reconstruction task with a denoise task of an autoencoder, and show that reconstructing images from cache activities aligns with denoising and reconstructing data, which can be done by an autoencoder. Through three ablation studies, we verify the hypothesis and demonstrate that an autoencoder performs significantly better than the Manifold-SCA framework. We further apply the autoencoder, implemented in the Manifold-SCA framework, to analyze practical cache activities collected by a profiling-based Prime+Probe attack [6]. Our results show that although an autoencoder can reconstruct partial pixel-related activities, they are insufficient to reconstruct images. The primary reason for this limitation is the huge pixel-dependent information loss in practical activities.

In summary, we make following contributions in this paper:

- Through the R+R (Reproducibility + Replicability) study, we show that the Manifold-SCA framework tends to generate images more than reconstruct them. This behavior deviates it from the goal of side-channel analysis, which aims to reconstruct existing secrets.
- With three ablation studies, we theoretically and experimentally show that an autoencoder is sufficient to reconstruct images from cache activities, which significantly outperforms the Manifold-SCA framework.
- We demonstrate that an autoencoder can reconstruct partial pixel-related activities from practical cache attacks, but these activities are insufficient to reconstruct images due to the information loss in the activities.

In this paper, we provide 10 reconstructed images and their reference images for each experiment in Figure 7. We release the artifact at our Github repo.

2. Background

In this section, we introduce background and related works on reconstructing images from cache activities.

Cache Attacks. Cache on modern x86 processors is a set-associative structure that stores recently accessed memory. Cache attacks [5, 27, 1, 6] are attacks that exploit the observation of cache activities to infer secrets from a victim process. L1D Prime+Probe attack targets the L1 data cache, which is the smallest but fastest cache in the memory hierarchy. It works by filling the cache with attacker's data and measuring the latency of accessing them. The attack used by Yuan et al. [12] is a profiling-based L1D Prime+Probe attack [6, 28]. It repeatedly measures the latency of accessing attacker's data to profile the entire L1D cache activities. We also use this attack to collect practical cache activities in our evaluation.

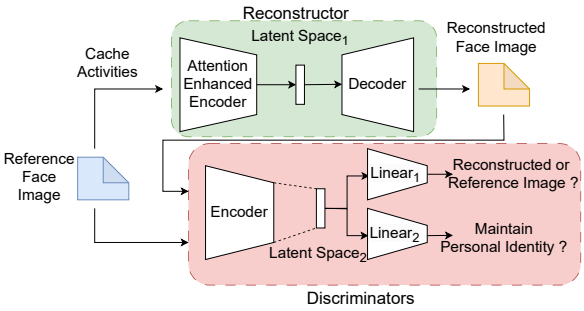

Figure 1. The Manifold-SCA framework overview.

Secret-dependent Cache Access. Cache attacks are powerful as they can break cryptographic algorithms that are mathematically secure [1, 29, 30, 31]. However, the attacks require the target to be implemented with secret-dependent cache accesses, which an attacker can observe and establish a relationship with the secrets. That is, after locating secret-dependent cache accesses, an attacker needs to build an algorithm to reconstruct secrets from them.

Cache Timing Attacks on Media Software. Compared to cryptographic libraries, media software is more complex and processes much larger inputs. Therefore, locating secret-dependent cache accesses within massive cache activities and connecting them with media inputs is challenging. Hähnel et al. [18] first demonstrated that an attacker could exploit cache leakages to reconstruct JPEG images under a compromised operating system. Specifically, an attacker single-steps the victim software to monitor cache accesses of pixel-dependent operations only. After collecting pixel-dependent cache activities, an attacker manually constructs an algorithm to map them back to pixel values. Although the collected activities are noise-less, it is still impossible to reconstruct images identical to their originals. This is because, during the transformation of pixel values to cache activities, multiple values are mapped to the same cache set. Consequently, an attacker can only reconstruct partial bits of a pixel byte from each pixel-dependent cache access.

The Manifold-SCA Framework for Image Reconstruction. Mapping pixel-dependent cache activities back to pixels is challenging, especially when an attacker is unfamiliar with the software. To systematically analyze side-channel activities to reconstruct media inputs, Yuan et al. [12] propose the Manifold-SCA framework, a deep-learning model, to automate the reconstruction procedure under a black-box scenario. The knowledge about victim software

is no longer required as the entire analysis is automated. Instead of focusing on the mapping from pixel values to cache activities, the framework operates under a hypothesis that the perception-level connection between cache activities and images can be presented in a joint manifold space. The "perception-level connection" refers to the relationship between high-level features of an image, such as hairstyle or gender, and cache activities, which can be learned by a deep learning model.

Figure 1 presents an overview of the Manifold-SCA framework. It consists a reconstructor, R, and two discriminators, D_{TF} and D_{ID}. The reconstructor is built with an autoencoder [20, 32, 21], which is enhanced by attention modules [33]. Two discriminators are built with linear classifiers: D_{TF} distinguishes whether an image is produced by R or not; D_{ID} identifies the identity of individuals in the images.

GAN-like Training Process. The Manifold-SCA framework is trained in a similar manner as generative adversarial networks (GANs) [24, 26, 22, 23, 24, 25], as the training of R is guided by D_{TF} and D_{ID}. The training process involves alternating updates to the reconsructor and discriminators. Initially, D_{TF} and D_{ID} are trained with images from the train set and images produced by untrained R. Next, R is trained with cache activities to output images that can pass the tests of D_{TF} and D_{ID}. The punishment for not passing the tests is converted to a loss function to update the networks of R. These two steps are repeated periodically until the performance of R is satisfactory.

Autoencoder in the Manifold-SCA Framework. Although the autoencoder in R is trained with two discriminators, its loss function also contains a reconstruction loss that measures the similarity distance between its output and input. We note that autoencoders are known for learning a representation of data and reconstructing data from the representation [20, 32]. They work by encoding a sample x to a hidden representation h using an encoder function f, and then decoding it back to a reconstructed sample $\hat{x}$ using a decoder function g. The goal is to find a hidden representation h that can map any sample x to $\hat{x}$. Due to its advantage in finding meaningful representations of data, autoencoders are being used to denoise data [32, 15] and reconstruct data [34].

Related Work. Hähnel et al. [18] first reconstructs images pixel-by-pixel from pixel-dependent cache activities. Yuan et al. [13] first applies a generative model to reconstruct images from instrumented cache activities. Yuan et al. [12] first proposes a general framework to analyze side-channel activities to reconstruct media inputs. Yeh and Sekiya [35] improves the Manifold-SCA framework to achieve better performance.

3. Manifold-SCA R+R Evaluation

In this section, we evaluate the reproducibility and replicability of the Manifold-SCA framework in reconstructing images from cache activities. We base our evaluation on its artifact, whose reproducibility has been validated by the USENIX Artifact Evaluation Committee. We show that the performance reported by the paper was overly optimistic due to a misinterpretation of the evaluation metric.

3.1. Evaluation Objective

The objective of our evaluation is to validate whether the Manifold-SCA framework can reconstruct images from cache activities with similar performance reported by the paper.

Source Images. The images to be reconstructed are sourced from the CelebA dataset [36], which comprises facial images of celebrities with 10,177 identities. Yuan et al. [12] crop each image to 128×128 pixels and force the face being positioned at the center.

Cache Activities. Two types of cache activities are analyzed to reconstruct images: instrumented and practical. Instrumented cache activities are collected using Intel Pin, a dynamic binary instrumentation tool. These activities contain cache accesses for each memory operation executed by the victim software. Practical cache activities are collected through a profiling-based Prime+Probe attack [6, 28], which obtains timing results through repeated priming and probing the L1D cache.

Evaluation Metric. Yuan et al. [12] utilize a commercial face comparison API, Face++ [37], to assess the framework's performance. The API returns a *confidence score* and three *confidence thresholds* to determine if two images are from the same person with three error rates (0.1%, 0.01% and 0.001%). Since Yuan et al. [12] claims to evaluate the performance with over 99.9% confidence scores [1], we believe the authors use the 0.1% error rate threshold. Namely, two images are considered from the same person if the confidence score is higher than the confidence threshold of 0.1% error rate.

TABLE 1. REPRODUCED RESULTS OF MATCHING RECONSTRUCTED IMAGES WITH THEIR REFERENCE IMAGES (MATCH / NON−FACE).

	Yuan et al. [12]	Misinterpreted Metric	Correct Metric
Instrumented	43.5% / 2.0%	41.7% / 1.5%	8.7% / 1.5%
Practical	36.9% / 0.8%	39.0% / 0.7%	10.6% / 0.7%

3.2. Reproducibility Evaluation

First, we introduce two inconsistencies between the paper and the artifact. Next, we correct them and reproduce the results.

Inconsistency: Undocumented Refiner. We find that the artifact always modifies the image produced by the Manifold-SCA framework with an external deep-learning model, a process not referenced in the paper. We name it Refiner as it visually refines framework's output to be more similar with those from the CelebA dataset. Although

1. Please see the first sentence on the right column of page 10 of [12].

the option of refining images can be "optionally" disabled through a flag, this feature is actually hardcoded to be always enabled (GitHub Link). We manually disable the Refiner for the evaluation to align with the paper's description of the framework. A detailed analysis of Refiner is provided in the Appendix A.

Inconsistency: Misinterpreted Evaluation Metric. Recall that the Face++ API returns a confidence score and three confidence thresholds after each query. We find that the artifact does not use any of the thresholds to determine if two images are from the same person. Instead, it uses a fixed threshold of 50 to interpret the confidence score (GitHub Link). In practice, the threshold of 0.1% error rate is always 62.327 for images provided by the artifact, which is higher than the fixed threshold utilized by the artifact. It means that more images would be considered from the same person with the fixed threshold than the actual threshold. Consequently, the performance reported by the artifact is likely to be higher than the actual performance. During the evaluation, we measured the framework's performance with both misinterpreted and corrected evaluation metrics.

Reproduced Results. We reproduce the results with all CelebA images, which is 1,000, and their corresponding instrumented and practical cache activities provided by the artifact. Since the artifact provides pre-trained models to analyze them, we do not train them from scratch. We report the face matching rate and non-face rate of the reconstructed images in Table 1. As illustrated in the table, we can reproduce the results consistent with the paper using the misinterpreted metric. However, after correcting it, the framework's performance is significantly reduced (79% and 73% respectively). Hence, we conclude that the results of analyzing cache activities to reconstruct images cannot be reproduced with the correct evaluation metric.

3.3. Replicability Evaluation

We replicate the results by re-collecting cache activities and training the framework from scratch. First, we introduce the procedure of replicating the results. Next, we present the replicated results.

Base Setting. When collecting cache activities, we disable Address Space Layout Randomization (ASLR) to prevent addresses being randomized. To align with the setting described in the paper, we build the train set and test set with 162,770 and 19,962 images.

Instrumented Activities Collection. We collect instrumented cache activities by instrumenting `libjpeg-turbo` library (version 2.5.1) with scripts provided by the artifact. We train the framework with script `recons_image.py` on an NVIDIA RTX A6000 GPU.

Practical Activities Collection. The results of practical cache attacks largely depend on hardware environment. To minimize the difference between our setting and the artifact's setting, we target the same `libjpeg-turbo` library used by the artifact. We collect practical cache activities with scripts provided by the artifact [2] on an Intel Core i7-9750H CPU, running Ubuntu 20.04 LTS. Since the artifact utilizes an empirical threshold to process the timing results, we also empirically determine the threshold on our platform. We documented the procedure in the Appendix B.

According to the instructions, we should train the model with script `pp_image.py`. However, we find this script implements two different network architectures: one similar to the one described in the paper and another is completely different. Specifically, the former network lacks an attention module, while the latter involves the training of Refiner. To align with the paper, we add the attention module to the former network and train it with instructions provided by the artifact. We train the model on an NVIDIA RTX 4070ti GPU.

Determine Well-trained Model. We note that Yuan et al. [12] did not mention how to determine if a model is well-trained. To ensure the results being evaluated are produced by the most well-trained model, we measure the structure similarity (SSIM) [38], a commonly used metric that measure the similarity between two images, between reconstructed images and their reference images (in the test set) after each epoch. We select the model with the highest average SSIM score as the well-trained model.

TABLE 2. REPLICATED RESULTS OF MATCHING RECONSTRUCTED IMAGES WITH THEIR REFERENCE IMAGES (MATCH / NON-FACE).

	Yuan et al. [12]	Misinterpreted Metric	Correct Metric
Instrumented	43.5% / 2.0%	39.3% / 1.2%	8.6% / 1.2%
Practical	36.9% / 0.8%	32.0% / 1.2%	5.2% / 1.2%

Replicated Results. We train two models with default hyperparameters and settings provided by the artifact. To be comparable with the reproduced results, we evaluate the performance using the same images that are used in the reproducibility evaluation. We use both misinterpreted and corrected evaluation metrics to measure the performance and present the results in Table 2.

According to the table, we are able to replicate results consistent with the reproduced results. However, after correcting the evaluation metric, the performance is significantly lower than the performance reported by the paper [12]. Therefore, we conclude that we cannot replicate the results of analyzing cache activities to reconstruct images with the correct evaluation metric.

4. Discussion: Reconstruction vs Generation

After reproducing and replicating the results, we find it hard to interpret if the Manifold-SCA framework reasonably analyzes cache activities, as evidenced by the low face-matching rate. Although the framework does produce images after the automatic analysis, it is unclear whether these images tend to be *reconstructed* or *generated*. Namely,

2. We made necessary but minimum changes to successfully run the script.

we cannot tell how much content of an image is inherited from its reference image and how much is generated by learning the distribution of the training set.

We note that it is essential to ensure ML-assisted SCA reconstructs images rather than generates new images. As the goal of side-channel analysis is to reconstruct existing data belonging to a victim, not to generate new data that does not exist in reality.

4.1. Distinguishability Evaluation

To help investigate whether the Manifold-SCA framework tends to reconstruct or generate images, we propose to evaluate the distinguishability of the framework's outputs. The reason is that a reconstructed image inherits features from its reference image, and these features are likely to be not shared with other images. Therefore, a reconstructed image should be more similar to its reference image while being less similar to other images in the dataset. In other words, the similarity between a reconstructed image and its reference image should be higher than that between it and other images. We name the difference in two similarities as *Distinguishability*. Hence, higher distinguishability on images denotes that the model tends to reconstruct images, while lower distinguishability on images denotes that it tends to generate images.

Distinguishability Evaluation Metric. To measure the distinguishability of an image, we compare the similarity between a reconstructed image and its reference image with the similarity between it and other images in the dataset. We use the success rate of identifying the reference image from a set of images as the distinguishability metric. We note that similar concepts have been used in previous ML-assisted SCA works [7, 8] that classify if a given side-channel activity is related to a specific website. The difference is that the comparison between similarities is implicitly conducted by deep-learning models (e.g. LSTM [39]) in these works, while we explicitly conduct the comparison in our evaluation. We formulate the distinguishability evaluation metric for evaluating reconstructed images as follows:

> Given a set IMG with N images, an img_i from it, which is unknown to the attacker, is processed by the victim. An attacker can reconstruct img'_i with the pre-trained ML-assisted SCA model after collecting the cache activities. Next, an attacker computes the similarity s_{ij} between img'_i and img_j for all images from IMG, with a similarity metric, such as SSIM [38]. To locate the image processed by the victim, an attacker selects the img_j with the highest s_{ij} as the guess candidate. We say img'_i is distinguishable from IMG if and only if $j = i$.

Distinguishability Evaluation Results. We evaluate the distinguishability of the replicated results with the first 10,000 images in the test set. Specifically, we build IMG by randomly selecting 10, 100, 200, and 500 images. We present the results in Table 3.

TABLE 3. REPLICATED RESULTS DISTINGUISHABILITY EVALUATION.

	$N = 10$	$N = 100$	$N = 200$	$N = 500$
Base Line	10.0%	1.0%	0.5%	0.2%
Instrumented	41.2%	13.3%	8.9%	5.3%
Practical	9.7%	1.0%	0.5%	0.3%

The baseline represents the outcome of randomly guessing the candidate. As shown in the table, the framework's performance decreases rapidly as the size of the image set (IMG) increases. Additionally, the performance of analyzing practical activities is close to the baseline, while the results for instrumented activities are slightly better than it. This suggests that the images produced by the framework have limited distinguishability from other images in the dataset. In other words, the framework tends to generate new images rather than reconstruct existing images. We note that since the results of analyzing instrumented activities are better than the baseline, it means that the Manifold-SCA framework has, but limited, ability to reconstruct images from cache activities.

Results Reasoning. We hypothesize that the framework's tendency to generate new images is a consequence of its architectural design. Recall that the framework is trained with discriminators in a manner similar to train generative adversarial networks (GANs) [26]. Since GANs are known for generating new data, it is possible that these two discriminators guide the framework to generate new images rather than reconstruct existing ones.

4.2. An Alternative View of Image Reconstruction

Cache attacks work by first observing the cache activities produced by the victim software and then building a relationship between them and secrets. That is, we can consider the cache activities $\tilde{x}$ as a transformation of secret inputs x, and the procedure can be written as $\tilde{x} = T(x)$. Since it is a transformation, $\tilde{x}$ is an interpretation of x. However, compared to x whose each value is a secret, $\tilde{x}$ contains noises that are not related to x. For instance, cache activities that are not generated by processing pixels, or impact from other processes running in parallel. Side-channel analysis is a procedure of reconstructing x from $\tilde{x}$. Being inspired by the autoencoder in the Manifold-SCA framework, we find this procedure is similar to the denoise task of autoencoder [32].

Autoencoder for Denoise. An autoencoder can be used to denoise data, which works as follows. Given an input data x and a corruption process C, the corrupted data is $\tilde{x} = C(x)$. It first extracts a representation $h_{\tilde{x}} = f(\tilde{x})$ from an encoding function f. Next, it tries to reconstruct it back to $x \approx g(h_{\tilde{x}})$ by a decoding function g. Its objective is to learn the probability distribution $p_{reconstruct}(x|\tilde{x})$ by $p_{decoder}(x|h(\tilde{x}))$. In other words, an autoencoder extracts a

meaningful representation for x from $\tilde{x}$ and decodes it back to x.

Autoencoder and SCA. We note that the denoise procedure is similar to side-channel analysis. The transformation process T can be seen as a corruption process C, which introduces noises but still contains features of the original data. In the context of reconstructing images from cache activities, the noises are cache activities that are not related to pixels, and the features are cache activities that are caused by processing pixels. Hence, the reconstruction process can be considered as a denoise task where the corruption process C is replaced by the pixel processing process T. Therefore, we hypothesize that an autoencoder is sufficient for reconstructing images from cache activities.

In the next section, we validate this hypothesis through three ablation studies on the Manifold-SCA framework. We show that a single autoencoder outperforms Manifold-SCA in reconstructing images that are more similar to their reference images but less similar to the others. We note that the autoencoder used in the Manifold-SCA framework is enhanced by an attention module. However, it still aligns with the definition of an autoencoder, which encodes data to a latent space and decodes it back to the original data to minimize the differences between the encoded and decoded data [32]. Therefore, we use autoencoder to represent the autoencoder implemented in the Manifold-SCA framework in the following sections.

5. Ablation Study

In this section, we conduct three ablation studies to show an autoencoder is sufficient to reconstruct images from instrumented cache activities, while the discriminators in the Manifold-SCA framework misguide it to generate new images.

5.1. Instrumented Cache Activities under White-box Scenario

To better explain the results of ablation studies, we eliminate uncertainties from analyzing cache activities caused by the entire image decompression process (black-box scenario) and focus on those caused by processing image pixels only (white-box scenario).

Previous works [19, 18] have demonstrated that pixel-related activities can be exploited to reconstruct images. Therefore, it is reasonable to assume that a deep-learning model can automate this procedure. The only question is how well each model can perform, which reflects the impact of each component in the framework.

In a nutshell, the leaky function is invoked 384 times to process an image of 128×128 pixels. 256 times are for the Y channel (black-white value) and 64 times each for the Cb (blue pixel) and Cr (red pixel) channels. Within each invocation, it processes 8x8 pixels by looking up the same table with pixel values. The function will skip the rest seven pixels of a row if they are the same as the first pixel. Therefore, the total number of pixel-related cache activities is variable, with a maximum of 24,576 (384×64). More information can be found in previous works [19, 18].

For the study, we collect pixel-related cache activities by instrumenting the leaky function in `libjpeg-turbo` library. We represent the activities of decompressing each image as a matrix of 384×64, where each row contains all accesses made by one invocation, and each value represents the accessed cache set. Since the L1D cache has 64 sets, we use values from 0 to 63 to denote cache sets accessed by the function and 64 to denote skipped accesses.

5.2. Decoupling the Manifold-SCA Framework

We decouple the Manifold-SCA framework into three components.

Reconstructor (R) The first component we evaluate is the reconstructor of the Manifold-SCA framework, which we refer to as R. It consists of a classic autoencoder (AE) model enhanced by a Convolutional Block Attention Module (CBAM) [33]. It functions as a classic autoencoder, which encodes cache activities to a latent space and decodes them into images. Adding attention mechanisms improves its performance by capturing spatial and channel-wise attention to the activities. We note that while CBAM affects the autoencoder's performance, it does not change the nature of the model. Following the loss function designed by the Manifold-SCA framework, we train R with Mean Squared Error (MSE) loss [40]:

$$L_R = MSE(IMG', IMG)$$

where IMG' is the output of the decoder and IMG is its reference image.

Reconstructor with D_{TF} Discriminator (R-D_{TF}). The second component we evaluate is the D_{TF} discriminator. It distinguishes whether an image is generated by the reconstructor or originates from the train set. Therefore, its output is binary, and we employ Binary Cross-Entropy (BCE) [41] loss to train it. To measure its influence, we construct a generative network with R and D_{TF} and refer to it as R-D_{TF}. Since the reconstructor needs to pass the test of D_{TF}, we build loss functions as follows:

$$L_{D_{TF}} = BCE(Predict, Label)$$
$$L_{G-D_{TF}} = 100 \cdot L_R + L_{D_{TF}}$$

where $predict$ is the prediction made by the discriminator and $Label$ is the ground truth.

Reconstructor with D_{ID} Discriminator (R-D_{ID}). The third component we evaluate is the D_{ID} discriminator. It predicts the identity of the individual in an image, constituting a multi-class classification problem. Hence, we employ Cross-Entropy (CE) [42] loss to train it. To access its impact, we construct a generative network with R and D_{ID}, denoted as R-D_{ID}. Similar to the loss function of R-D_{TF}, we build loss functions as follows:

$$L_{D_{ID}} = CE(Prediction, Label)$$
$$L_{G-D_{ID}} = 100 \cdot L_R + L_{D_{ID}}$$

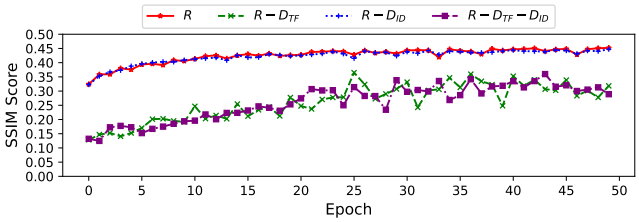

Figure 2. Performance variations after each epoch in the first ablation study.

TABLE 4. COMPREHENSIVE EVALUATION FOR THE FIRST ABLATION STUDY.

R	D_{TF}	D_{ID}	Analysis			Δ		
			$SSIM$	$Face++$	DIS	$SSIM$	$Face++$	DIS
✓	✓	✓	0.37	43.8	94.4	-	-	-
✓	✓		0.36	37.2	91.3	0.01↓	6.6↓	3.1↓
✓		✓	0.45	65.5	96.7	0.08↑	5.8↑	2.3↑
✓			**0.45**	**67.7**	**97.0**	**0.08↑**	**7.9↑**	**2.6↑**

$SSIM$ represents the average SSIM score of the test set.
$Face++$ represents the face matching rate (%).
DIS represents the success rate (%) of distinguishability evaluation.
Δ represents the change in performance compared to the baseline.

5.3. Study 1: Reconstruction or Generation

In the first ablation study, we aim to understand the impact of each component on the framework's output, particularly in terms of either reconstructing or generating images from cache activities.

Experiment Design. We decouple the Manifold-SCA framework into three models: R, R-D_{TF} and R-D_{ID}, and train them to reconstruct images from instrumented cache activities under a white-box scenario. For comparison, we also train the full Manifold-SCA framework (R-D_{TF}-D_{ID}) to analyze the same cache activities.

Dataset. We construct the train and test sets with images from the CelebA dataset [36] and their corresponding cache activities, formatted as matrices of 384×64. The training set comprises 162,770 images and their associated cache activities, while the test set consists of 19,962 images and their cache activities. Each image is cropped to center the faces and resized to 128×128 pixels.

Model Training. We train models on an NVIDIA RTX A6000 GPU with 48GB memory. Each experiment runs for 50 epochs, with performance evaluations on the test set conducted after each epoch. To determine the optimal training point, we measure the SSIM score between each output and its corresponding reference image, selecting the model from the epoch that achieves the highest average SSIM score as the final model. We leave mode details about model implementations and training process in Appendix C.1.

Results Overview. We present the SSIM score variations over the number of epochs in Figure 2. According to the figure, models incorporating D_{TF} consistently perform worse than those without it. Meanwhile, $R+D_{ID}$ performs similarly to R. These suggest that D_{TF} negatively impacts the framework by producing images less similar to the references, while D_{ID} has limited impact on the results.

Comprehensive Evaluation. We now conduct a comprehensive performance evaluation using three metrics: SSIM score, face matching rate (with Face++), and distinguishability rate. We use previous two metrics to evaluate the similarity between reconstructed images and their references, and the last metric to evaluate the distinguishability of them. Overall, we evaluate the first 10,000 images from the test set with the selected final models. The results are presented in Table 4. First, we observe that R achieves the best performance across all metrics. This indicates that an autoencoder, R, is sufficient to reconstruct images that are similar to the reference ones but different from others. Second, the performance of $R + D_{ID}$ is similar to that of R, proving that the D_{ID} component is not essential. In the end, the inclusion of D_{TF} significantly degrades model's performance across all metrics. This demonstrates that D_{TF} substantially affects the training of the autoencoder, guiding it to generate images that are less similar to corresponding references.

Discussion. We notice that all models perform well, with some variance in their performance, in the distinguishability analysis. This indicates that they are all capable of analyzing cache activities to reconstruct images. This is expected, as each model includes an autoencoder (R), which effectively compresses inputs (cache activities) into meaningful representations (pixel-related information) and decodes them back to images. However, the addition of discriminators affects the training of the autoencoder. Since the autoencoder needs to compete discriminators by approximating the training data distribution, it deviates from merely reconstructing images to generating images that can satisfy the discriminators.

We note that since the analyzed cache activities only contain pixel-related information, it is relatively easy for a deep-learning model to learn from them and reconstruct images. Therefore, the performance improvement by removing discriminators is limited in this context. However, as we will show in the third ablation study, removing discriminators can significantly improve the performance in more challenging tasks.

5.4. Study 2: Performance Improvement

After confirming that an autoencoder is sufficient to reconstruct images, we conduct the second ablation study to discuss potential ways to improve its performance. Specifically, we evaluate two modifications proposed by a follow-up work on the Manifold-SCA framework [35].

Two Changes of the Manifold-SCA Framework. Yeh and Sekiya [35] claim to achieve better performance in reconstructing images by making two changes to the Manifold-SCA framework. First, the authors replace the 2D convolutional layers in the encoder of the autoencoder with 1D convolutional layers. 2D convolutional layers use an $h \times w$ filter, where $h > 1$, to extract features from an input, while 1D convolutional layers use a $1 \times w$ filter to extract

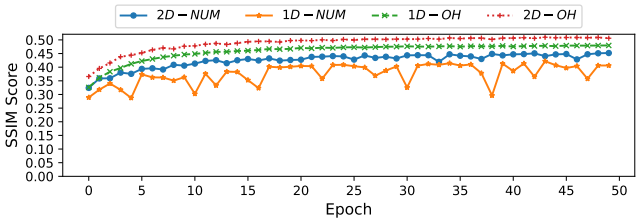

Figure 3. Performance variations after each epoch in the second ablation study.

features from an input. Second, the authors preprocess cache activities using one-hot encoding, instead of representing them as numeric data. That is, each cache index is treated as categorical data and converted to a binary vector of 64 values. If a cache set is accessed, the corresponding value in the vector is set to one, while the rest are set to zero. For instance, an access to the first cache set, typically represented as zero in numeric data, would be represented as [1, 0, 0, ..., 0] in one-hot-encoded data. While the authors claim both changes improve the performance of the Manifold-SCA framework, concrete evidence to support these claims are not provided.

Experiment Design. To apply the first change, we follow Yeh and Sekiya [35]'s work to implement 1D convolutional layers in the encoder. To apply the second change, we one-hot encode the cache activities used in the first ablation study. We satisfy two changes by implementing four different reconstructors: R_{2d-num}, R_{1d-num}, R_{2d-oh}, and R_{1d-oh}. Therefore, we have four experiments in this study: ❶ Analyze numeric data with R_{2d-num}, ❷ Analyze numeric data with R_{1d-num}, ❸ Analyze one-hot-encoded data with R_{2d-oh}, and ❹ Analyze one-hot-encoded data with R_{1d-oh}. Since experiment ❶ has been done in the first ablation study, we reuse its results as the baseline.

Dataset. We reuse the dataset being used in the first ablation study. In addition, we preprocess cache activities with one-hot encoding for the last two experiments. This preprocessing enlarges the size of each input by 64 times, as each numeric value is converted to a binary vector of 64 values. Consequently, we shape each cache activity as a matrix of 24576×64, whereas it used to be 384×64.

Model Training. We detail the implementations of four reconstructors in the Appendix C.2. Same to the first ablation study, we train each model 50 epochs and select the model with the highest average SSIM score as the final model.

Results Overview. We present the performance changes with the number of epochs in Figure 3. As shown in the figure, the autoencoder analyzes one-hot-encoded data better than numeric data, regardless of the encoder design. However, using 1D convolutional layers in the encoder decreases the performance for both representations of activities, especially for numeric data.

Comprehensive Evaluation. We present the comprehensive evaluation results in Table 5. According to the table, we observe that the autoencoder performs best with 2D convolutional layers and one-hot-encoded activities in all metrics. This indicates that images reconstructed under this setting have the highest similarity to the reference images and are the most distinguishable from the others. Conversely, we confirm that using 1D convolutional layers does not improve performance for either activity representations. However, since R_{1d-oh} (at 3^{nd} row) performs better than R_{2d-num} (at 2^{nd} row), we conclude that one-hot encoding activities can compensate the drawback of using 1D convolutional layers.

TABLE 5. COMPREHENSIVE EVALUATION FOR THE SECOND ABLATION STUDY.

Encoder		Activity		Analysis			Δ		
1D	*2D*	*NUM*	*OH*	*SSIM*	*Face++*	*DIS*	*SSIM*	*Face++*	*DIS*
	✓	✓		0.45	67.7	96.9	-	-	-
✓		✓		0.44	54.9	94.6	0.01↓	12.8↓	2.3↓
✓			✓	0.48	74.5	99.1	0.03↑	6.8↑	2.2↑
	✓		✓	**0.51**	**80.5**	**99.9**	**0.06↑**	**12.8↑**	**3.0↑**

Discussion. Through this ablation study, we demonstrate that proper preprocessing cache activities can significantly improve the performance of ML-assisted SCA. According to Yeh and Sekiya [35]'s explanation, one-hot encoding activities shift the model's focus from numerical differences to categorical differences. That is, accessing the first cache set and the last cache set (0 vs 63 in numeric) are treated equally with one-hot encoded data, as they just represent access to different categories (cache sets). However, our experimental results do not support their explanation regarding the advantage of using 1D convolutional layers. We believe the contribution of this ablation study goes beyond improving the performance of a model. More importantly, we emphasize that despite having "theoretical" explanations, experimental evidence is essential in designing and evaluating ML-assisted SCA methodologies.

5.5. Ablation Study 3: Autoencoder under Black-box Scenario

So far, we have verified that an autoencoder is sufficient to reconstruct images from pixel-related cache activities. However, its advantage over the Manifold-SCA framework is not apparent due to the simplicity of white-box tasks. In this study, we demonstrate the advantage of an autoencoder under a black-box scenario, which is a more challenging task than the white-box scenario.

Experiment Design. To fairly compare the performance of a single autoencoder with the Manifold-SCA framework, we reuse the implementations from previous works [12, 35]. Specifically, we analyze numeric data with the Manifold-SCA framework implemented by Yuan et al. [12] and one-hot-encoded data with the framework implemented by Yeh and Sekiya [35]. We build two autoencoder models by simply removing the discriminators from their implementations. Furthermore, we align the loss function of models by adopting the one designed by Yeh and Sekiya [35], which uses a combination of MSE and SSIM to measure the similarity between reconstructed images and reference

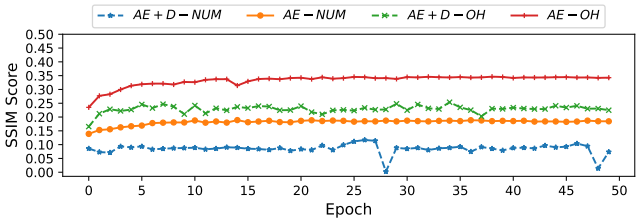

Figure 4. Performance variations after each epoch in the third ablation study.

images. We note that this combination has been proven to function better than using MSE or SSIM alone by previous works [43, 44].

Dataset. We build the train and test sets by following the setting of Yeh and Sekiya [35]'s work. Specifically, the train set contains $80,000$ images, and the test set contains $19,962$ images. We collect L1D cache activities for decompressing each image by instrumenting the `libjpeg-turbo` library. Since the number of cache activities for each image is not constant, we pad the activities with zeros to standardize their size. For numeric data, we keep the shape as $6 \times 256 \times 256$, consistent with the configuration used in Yuan et al. [12]'s work. For one-hot-encoded data, we keep the shape as $300,000 \times 64$, consistent with the configuration used in Yeh and Sekiya [35]'s work.

Model Training. We still train the models on an NVIDIA RTX A6000 GPU, but reduce the batch size from 100 to 50 due to the incredibly large size of one-hot-encoded data. Training each epoch for analyzing one-hot-encoded data takes around 50 minutes and occupying more than 30 GB of GPU memory. We leave detailed information about model implementations in Appendix C.3.

Results Overview. We present the performance changes across epochs in Figure 4. We use AE to represent a single autoencoder and AE+D to represent an autoencoder with discriminators, which corresponds to the Manifold-SCA framework. NUM and OH denote numeric data and one-hot-encoded data, respectively. As shown in the figure, the performance gap between an autoencoder and the Manifold-SCA framework is more pronounced in the black-box scenario than in the white-box scenario. It indicates that the autoencoder produces images more similar to the reference images than the Manifold-SCA framework does. Additionally, we observe a significant performance improvement when using one-hot-encoded activities. This observation reinforces the conclusion drawn from the second ablation study that one-hot-encoding activities improve the performance of the autoencoder.

Comprehensive Evaluation. We present the comprehensive evaluation results in Table 6. The baseline is set using the results of analyzing numeric data with the Manifold-SCA framework, as this was the initial general framework proposed for analyzing cache activities [12]. First, we observe that Manifold-SCA cannot produce distinguishable images with $80,000$ training images as its distinguishability evaluation result is close to a random guess (0.2%). This

TABLE 6. COMPREHENSIVE EVALUATION FOR THE THIRD ABLATION STUDY.

Model		Activity		Analysis			Δ		
AE+D	AE	NUM	OH	$SSIM$	$Face++$	DIS	$SSIM$	$Face++$	DIS
✓		✓		0.12	7.9	0.3	-	-	-
	✓	✓		0.19	15.2	4.4	0.07↑	7.3↑	4.1↑
✓			✓	0.25	30.5	55.1	0.13↑	22.6↑	54.8↑
	✓		✓	**0.35**	**41.7**	**74.2**	**0.23↑**	**33.8↑**	**73.9↑**

could be because the amount of training data is insufficient to train the model. In our replicability evaluation, we used 162,770 training images, while here, we use 80,000 training images. However, as shown in the second row, the performance of analyzing numeric data is improved around twice by using an autoencoder alone. This indicates that the Manifold-SCA framework is harder to train and performs worse than an autoencoder. Second, we find that while the Manifold-SCA framework can reconstruct images from one-hot-encoded data, its performance is still inferior to that of an autoencoder. Specifically, the distinguishability of images is increased by 34.7% when using an autoencoder alone. In summary, we prove that a single autoencoder can reconstruct images more similar to reference images while being less similar to others compared to the Manifold-SCA framework. It confirms that an autoencoder is sufficient to analyze cache activities to reconstruct images.

5.6. Discussion

Through three ablation studies, we demonstrate that a single autoencoder outperforms the Manifold-SCA framework in reconstructing images not only from pixel-related activities but also from more complex cache activities. This aligns with our discussion in Section 4.2, where we anticipated such results. Since pixel-related cache activities are a transformation of pixel values, it is expected that an autoencoder can find a meaningful representation to convert them back to images. In a black-box scenario, instrumented cache activities contain activities that do not depend on pixel values, which can be considered as noises. Here, the autoencoder functions as a denoiser, effectively removing the noise and reconstructing images from the pixel-related activities. While we only demonstrate the advantage of the autoencoder in the context of reconstructing images from cache activities, we hypothesize that it can be applied to most SCA scenarios where side-channel activities contain enough secret-dependent information (pixel, in this case).

6. Analyze Practical Cache Activities

After showing that an autoencoder is sufficient to reconstruct images from instrumented cache activities, we now investigate how it performs in analyzing practical cache activities. Compared to instrumented cache activities, practical cache activities are more complex and noisier. Therefore, we first evaluate how an autoencoder denoises and extracts pixel-related cache activities from practical cache activities.

Then, we evaluate whether the extracted activities are sufficient to reconstruct images. Our results show that the autoencoder can reconstruct partial pixel-related cache activities, but these are insufficient to reconstruct images. However, we find that the distinguishability of the reconstructed activities is significantly higher than the reconstructed images.

6.1. Statistical Analysis of Practical Cache Activities

Before analyzing practical cache activities with an autoencoder, we first statistically analyze them to demonstrate the difference between them and instrumented cache activities. Specifically, we show that practical activities cannot capture each pixel-related cache access separately, which we describe it as information loss.

Collection Results. We collect cache activities, with a profiling-based Prime+Probe attack, for decompressing images in the train set (162,770 images) and the test set (19,962 images) on an Intel Core i7-9750H CPU. Each phase (prime/probe) measures the latency of accessing 64 L1D cache sets separately and returns a vector of 64 values. Thus, each vector represents the entire L1D cache activities between two phases. The number of collected vectors for decompressing one image varies from 302 to 700, while the average is 476 and the standard deviation is 17.9.

Results Processing. We do not perform any post-processing on the collected activities except padding the number of vectors to 800 with 0s to simplify the autoencoder's implementation. We keep cache activities as raw data. Yuan et al. [12] amplifies the latency of cache misses to facilitate the classification of cache hits and misses. We find that such an approach slows down the attack resolution, resulting in capturing fewer cache activities. We support this claim by computing the number of vectors returned by the tool of Yuan et al. [12]. For decompressing one image, their tool returns an average of 266 vectors, which is even fewer than the lowest number of vectors returned by our attack.

Furthermore, the amplification reduces the correlation between the number of collected activities and the number of pixel-related cache accesses. We compute the correlation using Pearson correlation coefficient, which ranges from -1 to 1. The results show that the correlation with our activities is 0.16, whereas the one computed with the amplified activities is 0.06. This indicates that our activities are more relevant to the changes in pixels among images. Therefore, we chose to not amplify the cache miss latency and keep the activities as raw data.

Information Loss. By analyzing the number of collected activities, we find that it is significantly fewer than the number of pixel-related cache accesses ($22,879$ on average). This suggests that each prime and probe phase captures multiple victim's cache accesses.

Recall that each phase measures the latency of accessing 64 cache sets and returns a vector of 64 values. Each vector only indicates whether a cache set is accessed by the victim (higher latency) or not (lower latency), but not the order of them. Therefore, the attack loses information regarding the order of the victim's cache accesses.

Furthermore, the attack does not capture the number of accesses to the same cache set between two phases. It is because only the first victim's access to a cache set evicts attacker's data, and subsequent accesses to the same cache set do not cause new eviction. We note that access latency depends on the number of evicted data, and without multiple evictions, the attack loses information on the number of victim' accesses to one cache set.

In summary, practical cache activities suffer from information loss as they do not capture the order of the victim's accesses and the number of accesses to a single cache set between each prime and probe phase.

6.2. Reconstructing Pixel-related Cache Activities

In this section, we analyze practical cache activities with the autoencoder. Similar to most cache attacks on cryptographic libraries [1, 29, 5, 27, 45, 46, 47], we assume that an attacker knows how `libjpeg-turbo` generates pixel-related cache activities. We first evaluate whether an autoencoder does aware of pixel-related cache accesses by identifying cache sets relevant to processing pixels. Next, we simplify the instrumented cache activities (used in the first ablation study) according to the information loss and evaluate the accuracy of reconstructing them from practical activities. Our results show that although an autoencoder does aware of pixel-related cache accesses, it has limited performance in reconstructing instrumented cache activities.

Model Output Design. For the first evaluation, the model's output is a vector of 64 values, where each value indicates the relevance of one cache set to processing pixels. The value is set to be one if the cache set is relevant and zero if it is not. For the second evaluation, we simplify the instrumented cache activities used in the first ablation study (Section 5.3) according to the information loss in practical cache activities. Recall that these activities were shaped as matrices of 384×64, where each row contains all accesses made by one invocation of the leaky function, and each value represents the index of accessed cache set. Since practical activities lose the order of accesses and the number of accesses to one cache set, we simplify the activities by removing the order and the number of accesses within each invocation. Thus, the simplified activities are still shaped as a matrix of 384×64, but each row now represents the cache status after executing the leaky function once. We set the corresponding value to one if a cache set is accessed, no matter how many times, during the invocation; otherwise, it is zero.

Results of Identifying Pixel-related Cache Sets. We update the autoencoder model used in the ablation studies to fit new input and output, and provide detailed information in Appendix D. We train our model on an RTX 4070ti GPU and observe that it converges within 15 epochs. Our evaluation shows that the autoencoder is able to identify all cache sets relevant to processing pixels with an accuracy of 68%. This indicates that the autoencoder has a basic

TABLE 7. DISTINGUISHABILITY EVALUATION RESULTS.

	$N = 10$	$N = 100$	$N = 200$	$N = 500$
Baseline	9.7%	1.0%	0.5%	0.3%
Reconstruct Instrumented Activities	**96.0%**	**71.5%**	**58.7%**	**41.7%**
Reconstructed Images	16.6%	2.7%	1.5%	0.7%

understanding of practical cache activities and can locate cache sets relevant to processing pixels.

Results of Reconstructing Simplified Instrumented Cache Activities. We now evaluate how well an autoencoder reconstructs simplified instrumented cache activities. Since its output is a matrix of 384×64, we evaluate performance by counting the number of rows that are exactly the same as those in the simplified instrumented cache activities. Our evaluation shows that the autoencoder is able to reconstruct cache activities with 34% of its rows being exactly the same as the ground truth. This indicates that an autoencoder can reconstruct partial pixel-related cache accesses from practical activities. However, the performance is limited, as more than half of the rows are not reconstructed correctly.

6.3. Reconstructing Images from Reconstructed Activities

After reconstructing simplified pixel-related cache accesses from practical activities, we now investigate whether they are sufficient to reconstruct images. To do so, we first train an autoencoder to learn the mapping between the simplified instrumented activities and pixels. Then, we utilize the pre-trained model to reconstruct images from the reconstructed activities.

The Pre-trained Model. The pre-trained model analyzes simplified instrumented cache activities to reconstruct images. Recall that the simplified instrumented cache activities omit the order of cache accesses and the number of accesses to one cache set, which contains less information than the instrumented cache activities used in the first ablation study. To our surprise, the model achieves an average SSIM score of 0.46 and a 98.5 success rate for distinguishability evaluation when $N = 500$. This indicates that the model is able to overcome the information loss in the simplified activities and reconstruct high-quality images. The excellent performance of the pre-trained model guarantees the reliability of the following evaluation. Namely, if the number of correctly reconstructed cache activities is sufficient to reconstruct images, the pre-trained model should be able to reconstruct images from them.

Results of Reconstructing Images. We utilize the pre-trained model to reconstruct images from the reconstructed activities. The results show that the pre-trained model is able to reconstruct images with an average SSIM score of 0.18. However, as we will discuss in the next section, the distinguishability between reconstructed images is extremely low, which indicates that the model fails to reconstruct images from practical cache activities.

6.4. Distinguishability Evaluation

We now present the results of distinguishability evaluation for the reconstructed activities and images. As a comparison, we also reconstruct images directly from the practical cache activities (black-box scenario), which achieves an average SSIM score of 0.21. We conduct distinguishability evaluation with various N for the results of three experiments:

- ❶ reconstructing pixel-related cache accesses from practical activities;
- ❷ reconstructing images from the reconstructed activities;
- ❸ reconstructing images directly from practical activities.

We use the replicated results of the Manifold-SCA framework as the baseline for the distinguishability evaluation and present the results in Table 7.

As shown in the table, although the SSIM score for reconstructing images directly from practical cache activities (0.21) is higher than the one achieved with the reconstructed activities (0.18), both of them achieve the same level of distinguishability when $N = 500$. Notably, the distinguishability of reconstructed activities is significantly higher than that of reconstructed images. This confirms that the autoencoder is able to reconstruct partial pixel-related cache accesses from practical activities. However, the amount of correctly reconstructed activities is insufficient to reconstruct images with distinguishable features.

Discussion. First, it is expected that the autoencoder can reconstruct partial pixel-related cache accesses, as the entire process can be considered a denoise procedure. Second, it is also reasonable that the reconstructed activities are not sufficient to reconstruct images. The lower bound of the image reconstruction performance is determined by the amount of pixel-related information in the cache activities. When too much pixel-related information is lost, it is extremely challenging to reasonably deduce the missing information from the remaining information. We note that the performance of ML-assisted SCA is not determined by how advance a machine-learning model is, but by how much secret-dependent information is contained in the side-channel activities.

7. Conclusion and Future Work

In this paper, we revisit the first ML-assisted SCA framework for analyzing media software with a case study of reconstructing images from cache activities. We first show that the introduction of a discriminator network in the framework misguides the framework to generate images rather than reconstruct them. We emphasize the importance of reconstructing secrets in ML-assisted SCA, as the goal of side-channel analysis is to reconstruct the victim's data but not to generate new data. We propose to not only measure the similarity between a reconstructed image and its reference image but also between it and other images. With

the new evaluation metric, we conduct ablation studies to demonstrate that an autoencoder is sufficient to reconstruct images from cache activities. This finding first confirms our hypothesis about side-channel analysis and the denoise task of an autoencoder. Second, it provides a new perspective on the design of ML-assisted SCA methodologies. We demonstrate that an SCA task can be converted to a classic machine learning task, which can be efficiently solved with well-developed machine learning theories and models. Finally, we investigate how an autoencoder performs in analyzing practical cache activities. We show that due to the information loss in practical cache activities, an autoencoder can only reconstruct partial pixel-related cache accesses, which are insufficient to reconstruct images. We believe our experience reported in this paper can benefit future research in ML-assisted SCA and side-channel analysis.

Future Work. In this paper, we have only analyzed the performance of an autoencoder in reconstructing images from cache activities. Although an autoencoder has the potential to analyze more activities for different side-channel analysis tasks, we have not yet proved its performance in other tasks. Therefore, we plan to apply autoencoder to other side-channel analysis tasks in the future. Furthermore, to facilitate the community's advance in ML-assisted SCA, it is important to build a benchmark dataset, including both instrumented and practical cache activities, so that researchers can evaluate their methodologies in a consistent manner. We also leave this as a future work.

References

[1] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: The case of AES," in *CT-RSA*, vol. 3860, 2006, pp. 1–20.

[2] P. Kocher, J. Horn, A. Fogh, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *S&P*, 2019, pp. 1–19.

[3] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *USENIX Security*, 2018, pp. 973–990.

[4] J. V. Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx, "Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution," in *USENIX Security*, 2018, pp. 991–1008.

[5] Y. Yarom and K. Falkner, "FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack," in *USENIX Security*, 2014, pp. 719–732.

[6] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *S&P*, 2015, pp. 605–622.

[7] A. Shusterman, L. Kang, Y. Haskal, Y. Meltser, P. Mittal, Y. Oren, and Y. Yarom, "Robust website fingerprinting through the cache occupancy channel," in *USENIX Security*, 2019, pp. 639–656.

[8] J. Cook, J. Drean, J. Behrens, and M. Yan, "There's always a bigger fish: a clarifying analysis of a machine-learning-assisted side-channel attack," in *ISCA*, 2022, pp. 204–217.

[9] A. Shusterman, A. Agarwal, S. O'Connell, D. Genkin, Y. Oren, and Y. Yarom, "Prime+probe 1, javascript 0: Overcoming browser-based side-channel defenses," in *USENIX Security*, 2021, pp. 2863–2880.

[10] R. Wang, M. Brisfors, and E. Dubrova, "A side-channel attack on a higher-order masked crystals-kyber implementation," in *ACNS*, 2024, pp. 301–324.

[11] E. Dubrova, K. Ngo, J. Gärtner, and R. Wang, "Breaking a fifth-order masked implementation of crystals-kyber by copy-paste," in *APKC*, 2023, pp. 10–20.

[12] Y. Yuan, Q. Pang, and S. Wang, "Automated side channel analysis of media software with manifold learning," in *USENIX Security*, 2022, pp. 4419–4436.

[13] Y. Yuan, S. Wang, and J. Zhang, "Private image reconstruction from system side channels using generative models," in *ICLR*, 2021.

[14] Y. Yuan, Z. Liu, and S. Wang, "Cacheql: Quantifying and localizing cache side-channel vulnerabilities in production software," in *USENIX Security*, 2023, pp. 2009–2026.

[15] L. Wu and S. Picek, "Remove some noise: On pre-processing of side-channel measurements with autoencoders," *CHES*, pp. 389–415, 2020.

[16] S. Picek, I. P. Samiotis, J. Kim, A. Heuser, S. Bhasin, and A. Legay, "On the performance of convolutional neural networks for side-channel analysis," in *SPACE*, 2018, pp. 157–176.

[17] J. Kim, S. Picek, A. Heuser, S. Bhasin, and A. Hanjalic, "Make some noise. unleashing the power of convolutional neural networks for profiled side-channel analysis," *CHES*, pp. 148–179, 2019.

[18] M. Hähnel, W. Cui, and M. Peinado, "High-resolution side channels for untrusted operating systems," in *USENIX ATC*, 2017, pp. 299–312.

[19] Y. Xu, W. Cui, and M. Peinado, "Controlled-channel attacks: Deterministic side channels for untrusted operating systems," in *S&P*, 2015, pp. 640–656.

[20] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning internal representations by error propagation," *Biometrika*, pp. 599–607, 1986.

[21] D. Bank, N. Koenigstein, and R. Giryes, "Autoencoders," *CoRR*, 2020.

[22] C. Lazarou, "Autoencoding generative adversarial networks," *CoRR*, 2020.

[23] B. Ma, X. Wang, H. Zhang, F. Li, and J. Dan, "CBAM-GAN: generative adversarial networks based on convolutional block attention module," in *ICAIS*, 2019.

[24] M. Mirza and S. Osindero, "Conditional generative adversarial nets," *CoRR*, 2014.

[25] A. Radford, L. Metz, and S. Chintala, "Unsupervised representation learning with deep convolutional generative adversarial networks," *CoRR*, 2015.

[26] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio, "Generative adversarial nets," in *NIPS*, 2014, pp. 2672–2680.
[27] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+flush: A fast and stealthy cache attack," in *DIMVA*, 2016, pp. 279–299.
[28] 0xAde1a1de, "Mastik," 2022. [Online]. Available: https://github.com/0xAde1a1de/Mastik
[29] E. Tromer, D. A. Osvik, and A. Shamir, "Efficient cache attacks on aes, and countermeasures," *Journal of Cryptology*, vol. 23, no. 1, pp. 37–71, 2010.
[30] Y. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Cross-vm side channels and their use to extract private keys," in *CCS*, 2012, pp. 305–316.
[31] D. Genkin, R. Poussier, R. Q. Sim, Y. Yarom, and Y. Zhao, "Cache vs. key-dependency: Side channeling an implementation of pilsung," *CHES*, pp. 231–255, 2020.
[32] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.
[33] S. Woo, J. Park, J. Lee, and I. S. Kweon, "CBAM: convolutional block attention module," in *ECCV*, 2018, pp. 3–19.
[34] V. Kovenko and I. Bogach, "A comprehensive study of autoencoders' applications related to images," in *IT&I Workshop*, 2020, pp. 43–54.
[35] S. Yeh and Y. Sekiya, "Cache side-channel attacks against black-box image processing software," in *WEBIST 2023*, 2023, pp. 578–584.
[36] Z. Liu, P. Luo, X. Wang, and X. Tang, "Deep learning face attributes in the wild," in *ICCV*, 2015.
[37] "Face++." [Online]. Available: https://www.faceplusplus.com/
[38] Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," *IEEE transactions on image processing*, vol. 13, no. 4, pp. 600–612, 2004.
[39] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[40] N. Khare, P. S. Thakur, P. Khanna, and A. Ojha, "Analysis of loss functions for image reconstruction using convolutional autoencoder," in *CVIP*, 2021, pp. 338–349.
[41] U. Ruby and V. Yendapalli, "Binary cross entropy with deep learning technique for image classification," *Int. J. Adv. Trends Comput. Sci. Eng*, vol. 9, no. 10, 2020.
[42] I. J. Good, "Rational decisions," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 14, no. 1, pp. 107–114, 1952.
[43] G. Pang, C. Shen, L. Cao, and A. van den Hengel, "Deep learning for anomaly detection: A review," *ACM Comput. Surv.*, vol. 54, no. 2, pp. 38:1–38:38, 2022.
[44] Y. Chen, Y. Tian, G. Pang, and G. Carneiro, "Deep one-class classification via interpolated gaussian descriptor," in *AAAI*, 2022, pp. 383–392.
[45] C. Disselkoen, D. Kohlbrenner, L. Porter, and D. M. Tullsen, "Prime+abort: A timer-free high-precision L3 cache attack using intel TSX," in *USENIX Security*, 2017, pp. 51–67.
[46] A. Purnal, F. Turan, and I. Verbauwhede, "Prime+scope: Overcoming the observer effect for high-precision cache contention attacks," in *CCS*, 2021, pp. 2906–2920.
[47] A. Moghimi, G. Irazoqui, and T. Eisenbarth, "Cachezoom: How SGX amplifies the power of cache attacks," in *CHES*, 2017, pp. 69–90.

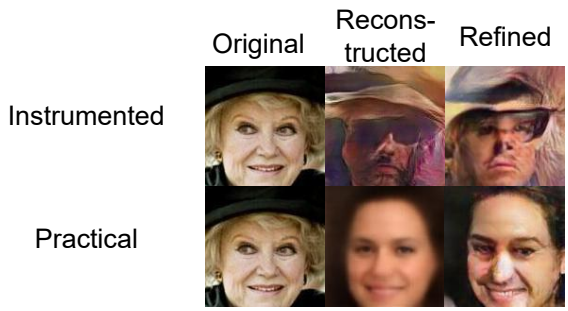

Figure 5. Visual changes made by the Refiner. *Reconstructed* represents the output of the Manifold-SCA framework. *Refined* represents the output of Refiner.

Appendix

1. The Undocumented Model: Refiner

The first inconsistency we identify is the utilization of an undocumented deep-learning model to enhance images generated by the framework. We denote this model as the *Refiner*, as it refines the framework's outputs. Although this model can be "optionally" disabled by setting a flag, we discover that it is hardcoded to always be enabled (GitHub Link). In other words, without modifying the source code, the undocumented model is inadvertently applied to refine the framework's results.

How It Works. The Refiner works similar as an AE-GAN model [22]. Its inputs are the reconstructed images produced from the Manifold-SCA framework. The Refiner then refines these images to produce outputs that deceive the discriminator into believing they are original, rather than artificially generated. Figure 5 shows two examples of the visual changes made by the Refiner, generated from data and pre-trained models provided by the artifact. As illustrated, the Refiner enhances images by reducing blurriness and adopting a style similar to that of the original images.

Reconstructed Features Preservation. In addition to altering the visual appearance of images, we observe that the Refiner also modifies the features of reconstructed images. Given that the evaluation dataset comprises face images of celebrities [36], the features present in the reconstructed images include facial attributes such as eyes and noses. These features can be abstracted to represent the identity of the person in the image. Notably, the Manifold-SCA

framework also leverages personal identity information to generate images.

We evaluate whether the Refiner preserves the features of reconstructed images by examining if the identity of the person in an image is changed. We use the same evaluation metric as Yuan et al. [12], which queries a commercial face comparison API, Face++ [37], with two images. The API returns a *confidence score* and three *confidence thresholds* to indicate if two images are from the same person with different error rates. We follow the paper [12] to use the 0.1% error rate threshold.

We conduct the evaluation by generating images from all cache activities provided by the artifact, totaling 1,000 for each type. We find that more than half of them have their identities changed after refinement. Specifically, only 41.7% of the images generated from instrumented activities and 46.4% of those generated from practical activities preserve their identities. This indicates that the Refiner is likely to alter the reconstructed features, leading to changes in the identity of an image.

2. Process Practical cache Activities

Practical cache activities are collected by a profiling-based L1D Prime+Probe attack [6], which fills the entire L1D cache (64 sets) with attacker's data and measures the time of accessing each of them. Therefore, each phase (prime/probe) returns a vector of 64 timing results, indicating the latency of revisiting each cache set. When a cache set is not accessed by the victim, the latency is relatively lower as the attacker's data remains in the L1D cache. Otherwise, the latency will be higher as the attacker needs to fetch data from lower-level caches, which costs more time. The former scenario is often referred to as a *cache hit* and the latter as a *cache miss*.

To find a threshold to distinguish two scenarios, we first need to know the distribution of the timing results. Therefore, we collect the data by profiling the cache activity of decompressing 1,000 CelebA images on an i7-9750H processor. The decompression is conducted by the libjpeg-turbo library 2.0.6, which is the same as the one used by the artifact.

We present the distribution of the timing results in Figure 6. According to the figure, the timing results are mainly distributed in two clusters, lower than 30 or higher than 60 cycles, indicating the distribution of cache hits and misses. Usually, the timing difference, which is known as L1D miss penalty, should be much smaller than 30 cycles. However, we notice that the customized Prime+Probe attack used by Yuan et al. [12] amplifies the cache miss. Specifically, it evicts the entire cache set, which contains eight cache lines, when only one cache line is evicted. Therefore, an attacker always has to fetch all eight cache lines from the lower-level cache, even if only one of them is evicted by the victim. It amplifies the cache miss penalty with roughly eight times, resulting in the two clusters in the figure.

Such an amplification does ease the procedure of finding the threshold, as the timing results are more distinguishable.

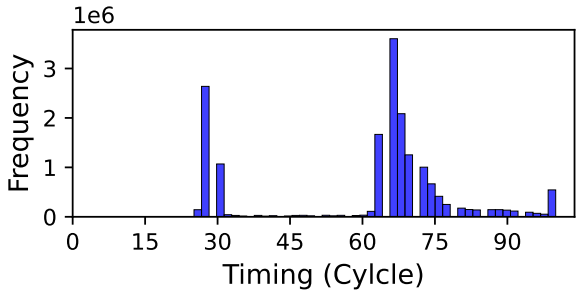

Figure 6. The timing distribution of priming and probing each cache set for decompressing 1,000 CelebA images.

TABLE 8. BASIC MODELS

Model	Layer
$basic\_2D\_conv$	Conv2d(in, out, $[k_x, k_y]$, $[s_x, s_y]$, pad) BatchNorm2d(out) LeakyReLU(0.2, inplace=True) CBAM(out)
$basic\_2D\_conv\_D_{enc}$	Conv2d(in, out, $[k_x, k_y]$, $[s_x, s_y]$, pad) BatchNorm2d(out) LeakyReLU(0.2, inplace=True)
$basic\_Resnet$	ReflectionPad2d(1) Conv2d(out, out, k=3, bias=False) BatchNorm2d(out) ReLU(True) ReflectionPad2d(1) Conv2d(out, out, k=3, bias=False) BatchNorm2d(out)
$basic\_2D\_convT$	ConvTranspose2d(in, out, k, s, pad) BatchNorm2d(out) ReLU(True) basic_Resnet(out)
$basic\_R_{dec}$	basic_2D_convT(128, 128, 4, 1, 0) basic_2D_convT(128, 128, 4, 2, 1) basic_2D_convT(128, 128, 4, 2, 1) basic_2D_convT(128, 128, 4, 2, 1) basic_2D_convT(128, 128, 4, 2, 1) ConvTranspose2d(128, 3, 4, 2, 1) Tanh()
$basic\_1D\_conv$	Conv1d(in, out, k) MaxPool1d(4) BatchNorm1d(out) LeakyReLU(0.2, inplace=True)
$basic\_D_{enc}$	Conv2d(3, 64, [4, 4], [2, 2], 1) LeakyReLU(0.2, inplace=True) $basic\_2D\_conv\_D_{enc}$(64, 128, [4, 4], [2, 2], 1) $basic\_2D\_conv\_D_{enc}$(128, 256, [4, 4], [2, 2], 1) $basic\_2D\_conv\_D_{enc}$(256, 512, [4, 4], [2, 2], 1) $basic\_2D\_conv\_D_{enc}$(512, 512, [4, 4], [2, 2], 1) Conv2d(512, 128, [4, 4], [1, 1], 0)
$basic\_D_{TF}$	Linear(128, 100) ReLU() Linear(128, 1) Sigmoid()
$basic\_D_{ID}$	Linear(128, 100) ReLU() Linear(128, 10177) Softmax(dim=1)

In the end, we empirically set the threshold to be 45 cycles, which is the middle point of the two clusters.

3. Architecture of Machine Learning Models and Trainings

Table 8 contains all basic components used in constructing models in ablation studies and practical cache activity analysis.

TABLE 9. MODELS ABLATION STUDY 1

Model	Layer
R_{study1}	basic_2D_conv(1, 16, [4, 4], [2, 2], 1) basic_2D_conv(16, 16, [4, 4], [2, 2], 1) basic_2D_conv(16, 32, [4, 4], [2, 2], 1) basic_2D_conv(32, 64, [4, 4], [2, 2], 1) basic_2D_conv(64, 128, [4, 3], [2, 1], 1) basic_2D_conv(128, 128, [4, 3], [2, 1], 1) Conv2d(128, 128, [6, 4], [2, 1], 1) Tanh() $basic\_R_{dec}$
D_{enc}	$basic\_D_{enc}$
D_{TF}	$basic\_D_{TF}$
D_{ID}	$basic\_D_{ID}$

3.1. Ablation Study 1. Optimizer. Adam with learning rate $2e-4$.

Train $R - D_{TF} - D_{ID}$**.** Same as Manifold-SCA [12].

Train $R - D_{TF}$**.** For each epoch, D_{TF} is trained first with original (IMG_o) and reconstructed (IMG_r) images respectively to correctly distinguish them.

$L = BCE(Predict_{IMG_o}, real)$ and $L = BCE(Predict_{IMG_r}, fake)$

Then R is trained with the reconstruction loss and the punishment loss if the image is identified as a reconstructed image by D_{TF}.

$L = 100 * MSE(IMG', IMG_r) + BCE(Predict_{IMG_r}, real)$

Train $R - D_{ID}$**.** For each epoch, D_{ID} is trained first with original (IMG_o) images respectively to correctly identify the individual's ID.

$L = CE(Predict_{IMG_o}, id)$

Then R is trained with the reconstruction loss and the punishment loss if the individual in the image is incorrectly identified by D_{ID}.

$L = 100 * MSE(IMG', IMG_r) + BCE(Predict_{IMG_r}, real)$

Train R**.** R is simply trained with the reconstruction loss.

$L = MSE(IMG', IMG_r)$

3.2. Ablation Study 2. Optimizer. Adam with learning rate $2e-4$.

Training. Since all of four models are single R, they are trained with the reconstruction loss only.

$L = MSE(IMG', IMG_r)$

3.3. Ablation Study 3. Optimizer. Adam with learning rate $2e-4$.

Train R_{ae-num} **and** R_{ae-oh}**.** Since both of them are single R, they are trained with the reconstruction loss only.

$L = MSE(IMG', IMG_r)$

Train $R_{ae-num} + D$ **and** $R_{ae-oh} + D$**.** Same as Manifold-SCA [12].

4. Practical Activities Experiments

Optimizer. Adam with learning rate $2e-4$.

Train $R_{trace2img}$ **and** $R_{reduced2img}$**.** Since both of them are single R, they are trained with the reconstruction loss only.

TABLE 10. MODELS ABLATION STUDY 2

Model	Layer
R_{1d-num}	basic_1D_conv(64, 64, 6) basic_1D_conv(64, 64, 4) basic_1D_conv(64, 128, 4) flatten() Linear(768, 128) Batchnorm1d(128) $basic\_R_{dec}$
R_{1d-oh}	basic_1D_conv(64, 64, 6) basic_1D_conv(64, 64, 4) basic_1D_conv(64, 128, 4) flatten() Linear(49152, 128) Batchnorm1d(128) $basic\_R_{dec}$
R_{2d-num}	basic_2D_conv(1, 16, [4, 4], [2, 2], 1) basic_2D_conv(16, 16, [4, 4], [2, 2], 1) basic_2D_conv(16, 32, [4, 4], [2, 2], 1) basic_2D_conv(32, 64, [4, 4], [2, 2], 1) basic_2D_conv(64, 128, [4, 3], [2, 1], 1) basic_2D_conv(128, 128, [4, 3], [2, 1], 1) Conv2d(128, 128, [6, 4], [2, 1], 0) Tanh() $basic\_R_{dec}$
R_{2d-oh}	basic_2D_conv(1, 16, [4, 4], [2, 2], 1) basic_2D_conv(16, 16, [4, 4], [2, 2], 1) basic_2D_conv(16, 32, [4, 4], [2, 2], 1) basic_2D_conv(32, 64, [4, 4], [2, 2], 1) basic_2D_conv(64, 128, [4, 3], [2, 1], 1) basic_2D_conv(128, 128, [4, 3], [2, 1], 1) basic_2D_conv(128, 128, [4, 4], [2, 2], 1) basic_2D_conv(128, 128, [4, 3], [2, 1], 1) basic_2D_conv(128, 128, [4, 3], [2, 1], 1) basic_2D_conv(128, 128, [4, 3], [2, 1], 1) Conv2d(128, 128, [6, 4], [1, 1], 0) Tanh() $basic\_R_{dec}$

TABLE 11. MODELS ABLATION STUDY 3

Model	Layer
R_{ae-num}	basic_2D_conv(1, 64, [4, 4], [2, 2], 1) basic_2D_conv(64, 64, [4, 4], [2, 2], 1) basic_2D_conv(64, 128, [4, 4], [2, 2], 1) basic_2D_conv(128, 256, [4, 4], [2, 2], 1) basic_2D_conv(256, 512, [4, 4], [2, 2], 1) basic_2D_conv(512, 512, [4, 4], [2, 2], 1) Conv2d(512, 128, [4, 4], [1, 1], 0) Tanh() $basic\_R_{dec}$
R_{ae-oh}	basic_1D_conv(64, 64, 6) basic_1D_conv(64, 64, 4) basic_1D_conv(64, 128, 4) basic_1D_conv(128, 256, 4) basic_1D_conv(256, 512, 4) basic_1D_conv(512, 512, 4) flatten() Linear(37500, 128) Batchnorm1d(128) $basic\_R_{dec}$
D_{enc}	$basic\_D_{enc}$
D_{TF}	$basic\_D_{TF}$
D_{ID}	$basic\_D_{ID}$

$L = MSE(IMG', IMG_r)$

Train $R_{trace2reduced}$**.** Since it is a single R, it is trained with the reconstruction loss only.

$L = BCE(Reduced', Reduced_r)$

Figure 7. The first 10 reconstructed images and their references for all experiments in the paper.

TABLE 12. MODELS PRACTICAL ACTIVITIES EXPERIMENTS

Model	Layer
$R_{trace2img}$	basic_2D_conv(1, 64, [4, 4], [2, 2], 1) basic_2D_conv(64, 64, [4, 4], [2, 2], 1) basic_2D_conv(64, 128, [4, 4], [2, 2], 1) basic_2D_conv(128, 256, [4, 4], [2, 2], 1) basic_2D_conv(256, 512, [4, 3], [2, 1], 1) basic_2D_conv(512, 512, [4, 3], [2, 1], 1) basic_2D_conv(512, 512, [5, 3], [2, 1], 1) Conv2d(512, 128, [5, 4], [1, 1], 0) Tanh() $basic\_R_{dec}$
$R_{trace2reduced}$	basic_2D_conv(1, 64, [4, 4], [2, 2], 1) basic_2D_conv(64, 64, [4, 4], [2, 2], 1) basic_2D_conv(64, 128, [4, 4], [2, 2], 1) basic_2D_conv(128, 256, [4, 4], [2, 2], 1) basic_2D_conv(256, 512, [4, 3], [2, 1], 1) basic_2D_conv(512, 512, [4, 3], [2, 1], 1) basic_2D_conv(512, 512, [5, 3], [2, 1], 1) Conv2d(512, 128, [5, 4], [1, 1], 0) Tanh() ConvTransposed2d(128, 128, 4, 1, 0) BatchNorm2d(128) ReLU(True) ConvTransposed2d(128, 384, 4, 2, 1) Sigmoid()
$R_{reduced2img}$	basic_2D_conv(1, 16, [4, 4], [2, 2], 1) basic_2D_conv(16, 16, [4, 4], [2, 2], 1) basic_2D_conv(16, 32, [4, 4], [2, 2], 1) basic_2D_conv(32, 64, [4, 4], [2, 2], 1) basic_2D_conv(64, 128, [4, 3], [2, 1], 1) basic_2D_conv(128, 128, [4, 3], [2, 1], 1) Conv2d(128, 128, [6, 4], [2, 1], 1) Tanh() $basic\_R_{dec}$