A. Implementation Details

Training. We set the number of perturbations K in MoP to four. When calculating MLP loss in Eq. 5, we use intermediate features from ["down_1", "down_2", "down_3", "mid_0"] layers in SD VAE, and set λ to 3.5×10^{-5}. We train FastProtect on a single A100 80GB GPU in an end-to-end manner except the assignment function $\mathcal{A}$. We use a batch size of 16, using the Adam optimizer [22] for 40k steps with learning rate of 0.0002, and betas of (0.5, 0.99). When training, the required VRAM and compute time are around 50GB and 12 hours, respectively. Note that since we update the perturbations with Adam instead of the direct optimization through PGD, we remove the minus sign in the training loss (Eq. 5) as: $\mathcal{L}_T(\mathbf{x}) = ||\mathbf{z} - \mathbf{z}_y||_2^2 + \frac{\lambda}{L}\sum_{l=1}^{L}||\mathcal{F}^l - \mathcal{F}_y^l||_2^2$. We utilize three patterned target images, each representing low, mid, and high pattern repetition, as illustrated in Figure 8. To implement adaptive targeted protection, for each protection strength (budget) η, we obtain three MoP models (low, mid, and high), each corresponding to one of the target images. The training algorithm is detailed in Algorithm 1; note that for simplicity, it is demonstrated with a batch size of one, but increasing the batch size is straightforward.

Inference. When an input image with resolution other than 512^2-px is given, we perform bilinear interpolation to the perturbations (δ_g, Δ) to match the resolution with the input image's. However, the SD VAE encoder receives downsampled image to a fixed 512^2-px to prevent a significant increase in computational load during VAE encoding. Although relative low-resolution is given to VAE, empirically, this shows robust performance to some extent. For adaptive targeted protection, we pre-compute and cache the average entropy of the target images. When an input image is given, we calculate its entropy and select the nearest target image based on the cached values. Hence, adaptive targeted protection incurs minimal overhead since FastProtect already obtains $\mathbf{z}$ in the MoP assignment stage. For the adaptive protection strength, we provide the surrogate protected image to LPIPS with its original resolution, since the LPIPS network (AlexNet) is significantly smaller than the VAE encoder thus it consumes negligible computational overhead.

When obtaining the final perceptual map in Eq. 7, we first apply min-max normalization to the spatial distance map calculated by LPIPS and then subtract it from one to reverse it. Subsequently, we adjust scale of the perceptual map with scale function $\mathcal{S}(\cdot)$. With a given perceptual map $\mathbf{M}$, which has resolution of $H \times W$, we first initialize scaled perceptual map $\mathbf{M}'$ as one, i.e., $\mathbf{M}' = \mathbf{1}^{H \times W}$. Then, we calculate the final perceptual map as follows:

$$\mathbf{M}'[\mathbf{M} < q_i] = \begin{cases} \beta^i \times \alpha & \text{if } i < c \\ \beta^i & \text{if } i \geq c \end{cases}, \tag{8}$$

Low repetition Mid repetition High repetition

Figure 8. Examples of target images used by FastProtect. In our target image analysis (Figure 3), we used both low and high repetition target images.

where $q_i = \text{decile}(\mathbf{M}, i)$. The function $\text{decile}(\mathbf{M}, i)$ computes the i-th decile value of $\mathbf{M}$ (from the highest value). This scaling function is designed to perform stepwise scaling to the perceptual map. For each region corresponding to a specific decile, we downscale by a factor of β. Without this scaling, many regions would take very small perturbations, significantly reducing the overall protection efficacy. For the areas up to the c-th decile, we additionally multiply by α to increase the perturbation intensity in the least noticeable regions. This compensates for the overall perturbation magnitude lost in more noticeable regions. In our work, we use $(\alpha, \beta, c) = (1.3, 0.91, 3)$. The inference algorithm is detailed in Algorithm 2.

B. Related Work

Our protection method involves injecting adversarial perturbations into input images, a method extensively studied in the adversarial attack domain. In this section, we discuss two areas closely related to our motivation.

Universal Adversarial Perturbation (UAP). Moosavi-Dezfooli et al. [36] demonstrated that a single perturbation could be used to attack various images, in contrast to the traditional image-specific perturbations. This approach is significantly more efficient in terms of computation time compared to per-instance adversarial attacks because it does not require iterative optimization for each individual image. Subsequently, Hayes and Danezis [14] presented a method for generating UAPs using generative models, showing improved efficiency and effectiveness over previous methods. Similarly, Mopuri et al. [38] utilized generative networks to create adversarial examples. In addition, there have been various attempts to find UAPs tailored to the image recognition tasks [5, 29, 37, 39]. However, these methods are not aligned with and cannot be adapted to the image protection task. Instead, in our study, we designed MoP with a focus on creating a protection framework that is both fast and maintains high performance. Our approach is specialized for image protection, differing from the general UAP-related methodologies introduced in adversarial attack field.

Invisible Adversarial Perturbation. Achieving high in-

Algorithm 1: Training stage of FastProtect (Single-batch example)

Input: Training dataset $\mathcal{X}_\mathcal{D}$, target images $\{\mathbf{y}^l, \mathbf{y}^m, \mathbf{y}^h\}$, # protection perturbations K, # VAE encoder layers L, Perturbation budget (strength) η, Training steps N

Output: Assignment function $\mathcal{A}$, mixture-of-perturbation $\{\delta_g, \Delta\}$

/* Compute assignment function prior to train MoP */

$\mathcal{A} \leftarrow \texttt{K-means++}(\mathcal{E}(\mathcal{X}_\mathcal{D}),\, K)$

/* MoP training */

Initialize $\{\delta_g,\, \Delta\} \leftarrow 0$

$\mathbf{z}_y^l,\, \mathcal{F}_y^l \leftarrow \mathcal{E}(\mathbf{y}^l,\, L)$; $\mathbf{z}_y^m,\, \mathcal{F}_y^m \leftarrow \mathcal{E}(\mathbf{y}^m,\, L)$; $\mathbf{z}_y^h,\, \mathcal{F}_y^h \leftarrow \mathcal{E}(\mathbf{y}^h,\, L)$ // Prepare target images

for $n \leftarrow 1$ **to** N **do**

$\mathbf{x} \leftarrow \texttt{sample-batch}(\mathcal{X}_D)$; $\mathbf{z} \leftarrow \mathcal{E}(\mathbf{x})$

$k \leftarrow \mathcal{A}(\mathbf{z})$; $t \leftarrow \arg\min_{i \in \{l,m,h\}} ||\mathcal{H}(\mathbf{z}) - \mathcal{H}(\mathbf{z}_y^i)||_1$ // Refer to Eq. 4 and 6

$\mathbf{x} \leftarrow \mathbf{x} + \delta_g^t + \Delta_k^t$

$\mathbf{z},\, \mathcal{F} \leftarrow \mathcal{E}(\mathbf{x},\, L)$

$\mathcal{L}_T = ||\mathbf{z} - \mathbf{z}_y^t||_2^2 + \frac{\lambda}{L} \sum_{l=1}^{L} ||\mathcal{F}^{t,l} - \mathcal{F}_y^{t,l}||_2^2$ // Multi-layer protection loss (Eq. 5)

$\{\delta_g^t,\, \Delta_k^t\} \leftarrow \texttt{Adam}(\nabla_{\{\delta_g^t, \Delta_k^t\}} \mathcal{L}_T)$

$\{\delta_g^t,\, \Delta_k^t\} \leftarrow \texttt{Clamp}(\{\delta_g^t,\, \Delta_k^t\},\, -\frac{\eta}{2},\, \frac{\eta}{2})$

end

Algorithm 2: Inference stage of FastProtect

Input: Image $\mathbf{x}$

Output: Protected image $\hat{\mathbf{x}}$

$H,\, W \leftarrow \texttt{size}(\mathbf{x})$

$\mathbf{z} \leftarrow \mathcal{E}(\texttt{resize}(\mathbf{x};\, (512, 512)))$

/* MoP with adaptive targeted protection */

$k \leftarrow \mathcal{A}(\mathbf{z})$

$t \leftarrow \arg\min_{i \in \{l,m,h\}} ||\mathcal{H}(\mathbf{z}) - \mathcal{H}(\mathbf{z}_y^i)||_1$ // Utilize pre-computed target entropy

$\delta \leftarrow \texttt{resize}(\delta_g^t + \Delta_k^t;\, (H, W))$ // Match MoP's resolution to input image

/* Adaptive protection strength */

$\mathbf{M} \leftarrow \texttt{LPIPS}(\mathbf{x},\, \mathbf{x} + \delta)$ // Use surrogate protected image

$\mathbf{M}' \leftarrow \mathcal{S}(1 - \mathbf{M})$ // Refer to Eq. 8

$\hat{\mathbf{x}} = \mathbf{x} + \mathbf{M}' * \delta$

Table 6. Quantitative report on the effect of adaptive targeted protection shown in Figure 7a.

Target Image	Inference Domains			
	Object	Face	Painting	Cartoon
Low rep.	200.1	**327.7**	347.5	**237.3**
Mid rep.	207.2	297.9	<u>349.3</u>	234.6
High rep.	<u>208.3</u>	270.7	348.5	211.8
Adaptive target	**208.5**	<u>320.2</u>	**349.4**	<u>235.4</u>

visibility in adversarial attacks has garnered significant research interest. Several methods focus on restricting the regions of perturbations. Some approaches use the L_0 norm to generate sparse perturbations [7, 35], while others confine perturbations to small, salient areas to reduce visual distortions [8]. Various studies have targeted specific elements such as low-frequency components [13, 32] and have utilized advanced constraints like color components [47, 59] or quality assessments [53]. Additionally, Luo et al. [31] explored techniques for creating invisible and robust adversarial examples based on the human visual system. More recently, Laidlaw et al. [25] investigated perceptual adversarial robustness and adopted LPIPS [58] in adversarial perturbation optimization.

Although invisibility has been highly emphasized in adversarial attacks, it is less explored in the image protection task. Many methods enhance invisibility implicitly by minimizing the budget used, but this often leads to a trade-off with protection efficacy, resulting in decreased perfor-

Table 7. Quantitative comparison when fix the perturbation strength (η) to eight for all the protection frameworks.

Method ($\eta = 8$)	Latency	Object	Face	Painting	Cartoon
	CPU / GPU	Invisibility: DISTS (↓) / Efficacy: FID (↑)			
AdvDM [28]	1210s / 35s	0.129 / 199.5	0.152 / 303.7	0.117 / 346.8	0.194 / 196.5
PhotoGuard [46]	370s / 7s	0.184 / 211.9	0.258 / **371.1**	0.165 / 377.4	0.221 / **227.6**
Anti-DB [51]	7278s / 225s	0.164 / 197.4	0.193 / 317.4	0.143 / 343.9	0.231 / 184.6
Mist [27]	1440s / 40s	0.185 / **217.2**	0.259 / 365.8	0.166 / **386.2**	0.223 / 223.7
Impasto [1]	830s / 19s	0.131 / 188.9	0.198 / 298.4	0.123 / 352.4	**0.179** / 201.1
SDST [56]	1410s / 24s	0.170 / 198.0	0.244 / 302.1	0.152 / 354.1	0.199 / 196.7
FastProtect	**2.9s / 0.04s**	**0.097** / 200.4	**0.149** / 308.9	**0.048** / 348.0	0.186 / 220.3

mance. Ahn et al. [1] constrain perturbations based on the human visual system using various modules during the optimization process. While effective, they require many ad-hoc components. In contrast, FastProtect uses LPIPS to create perceptual maps and apply masking, which is a simpler and more streamlined approach. This differs from Zhang et al. [58], who use LPIPS in the optimization process.

C. Experimental Setups

Datasets. We utilize the *Object*, *Face*, *Painting*, and *Cartoon* domains in both the training and benchmark datasets. When constructing the training dataset, we randomly sample 20k images from ImageNet [9] for the Object domain, resizing all images to 512^2. No augmentation is applied. For the Face domain, we use 20k images randomly sampled from the FFHQ dataset [19]. The images, originally 1024^2 and face-aligned [20], are all downscaled to 512^2, without further augmentations. The Painting dataset is created by randomly sampling 20k images from the WikiArt dataset [45], resizing them to 512^2. We filter out images with resolutions outside the range of 512 to 2048-px. For the Cartoon dataset, we use Webtoon artworks published on NAVER Webtoon, sampling 20k images only the works with permission from the original creators for research purposes. Similar to the Painting dataset, we apply resolution filtering and resize the images to 512^2.

To make the benchmark dataset, we select 20 objects from the personalization subject dataset proposed by Ruiz et al. [44] for the Object domain. Each object consists of 5-6 images. When we conduct the main comparison (Table 2), we use images resized to 512^2, and for the arbitrary image analysis (Table 4), we apply protection to the original resolution images. For the Face domain, we followe Van Le et al. [51] and randomly sample 20 identities from VGGFace2 [4], each identity consisting of 12 images resized to 512^2. The Painting dataset is compiled by randomly sampling (but not overlap to the training dataset) 20 artists from the WikiArt dataset, each artist represented by 10 artworks, all resized to 512^2. For the Cartoon domain, we randomly sample 20 Webtoon works (but also no overlap to the training dataset) from the Webtoon dataset. In this dataset, we only use facial images of major cartoon characters by cropping and aligning all the images. This reflects the common practice in cartoons and illustrations where characters are the main focus to both readers and artists. In the main comparison we use 512^2 images, while the arbitrary analysis maintains the original resolution.

The cartoon images used in the benchmark dataset are also permitted by the artists for research only purpose. The list of cartoon works featured in this paper is as follows: *<Yumi's Cells>*, *<Maru is a Puppy>*, *<Free Draw>*, *<The Shape of Nightmare>*, *<See You in My 19th Life>*, *<Lookism>*, and *<Love Revolution>*.

Baselines. FastProtect is compared with existing diffusion-based mimicry protection frameworks: AdvDM [28], PhotoGuard [46], Anti-DreamBooth (Anti-DB) [51], Mist [27], Impasto [1], and SDST [56]. We follow the official settings for all the baselines by using their official codes. However, for PhotoGuard, we use the target image of Mist [27] by following the protocols of Xue et al. [56] while we use the Impasto's target image for the Impasto.

Evaluation. For the primary evaluation assessment, we use DISTS [10] to measure the invisibility of the protected image and FID [15] to evaluate the protection efficacy of the personalized diffusion methods. In addition, in here, we also compare the protection frameworks using other metrics: To evaluate invisibility, we include $\text{LPIPS}_{\text{VGG}}$ [58] and AHIQ [26]. Note that $\text{LPIPS}_{\text{VGG}}$ measures the perceptual distance in the VGG feature space, which is different from the feature space used in our perceptual map creation module (*i.e.* AlexNet). For evaluating protection efficacy, we use TOPIQ-NR [6] and QAlign [54].

When preparing mimicry outputs, we personalize the diffusion models using the protected image with LoRA [17] by default, employing default settings for all the personalization methods. To generate outputs, we use different inference prompts than those used during training to simulate black-box caption scenarios [51]. For example, we train the diffusion models with LoRA using captions of "A painting in *<sks>* style" prompt and perform inference with "A painting of a house in *<sks>* style".

Table 8. Additional quantitative comparison on the Subject domain.

Domain: Subject	Invisibility			Protection Efficacy		
	DISTS (↓)	$\text{LPIPS}_{\text{VGG}}$ (↓)	AHIQ (↑)	FID (↑)	TOPIQ-NR (↓)	QAlign (↓)
AdvDM [28]	0.197	0.362	0.540	<u>220.0</u>	<u>0.458</u>	**2.139**
PhotoGuard [46]	0.203	0.347	0.575	**223.0**	0.506	2.396
Mist [27]	<u>0.185</u>	<u>0.322</u>	0.578	217.2	0.523	2.328
SDST [56]	0.242	0.402	0.575	219.2	0.542	2.442
Anti-DB [51]	0.239	0.413	0.510	214.4	**0.439**	<u>2.148</u>
Impasto [1]	0.201	0.378	**0.588**	213.8	0.473	2.183
FastProtect	**0.155**	**0.258**	<u>0.583</u>	**223.0**	0.507	2.364

Table 9. Additional quantitative comparison on the Face domain.

Domain: Face	Invisibility			Protection Efficacy		
	DISTS (↓)	$\text{LPIPS}_{\text{VGG}}$ (↓)	AHIQ (↑)	FID (↑)	TOPIQ-NR (↓)	QAlign (↓)
AdvDM [28]	0.173	0.338	0.528	303.8	**0.466**	<u>2.493</u>
PhotoGuard [46]	0.189	0.357	0.557	<u>308.7</u>	0.630	3.113
Mist [27]	<u>0.154</u>	0.310	0.560	307.5	0.667	3.131
SDST [56]	0.244	0.431	0.561	302.1	0.629	3.019
Anti-DB [51]	0.162	<u>0.309</u>	0.543	301.4	<u>0.484</u>	**2.411**
Impasto [1]	0.198	0.388	<u>0.562</u>	298.4	0.565	2.939
FastProtect	**0.149**	**0.280**	**0.566**	**308.9**	0.558	2.851

D. Discussions

Arbitrary Resolution. As shown in Table 4, FastProtect can effectively handle arbitrary resolution images simply by resizing the MoP to match the resolution of an input image. In contrast, the baseline method, PhotoGuard [46], suffers a much greater performance drop compared to ours. There are two main factors that can impact protection performance for arbitrary resolution scenarios. The first factor is the need to adjust the size of the perturbation to match the size of the image when injecting the perturbation. In theory, iterative optimization methods do not face this issue. However, in practice, due to the VRAM constraint of GPUs, images beyond (typically 1024^2-px in a A100) a certain size must be split and protected in segments. On the other hand, our model does not have memory issues, but since the pre-trained MoP is fixed as 512^2-px, it requires resizing to fit an input image. The second factor is the necessity to downsize the protected image due to resolution requirements that each diffusion models has (e.g., 512^2 for SD v1, 768^2 for SD v2, or 1024^2 for SD-XL).

Our method appears to be affected by the information loss due to the two resize operations (one when applying MoP and the other during personalization). Of course, we observed that all protection frameworks suffer some degree of protection efficacy reduction during personalization, and our model is no exception (e.g., FID: 220.3 to 204.0). Such a reduction is due to the fine perturbations being lost during the downsize process (to match the resolution that of the diffusion models require). In contrast, although PhotoGuard performs optimization in a full-resolution, its robustness in this aspect seems limited. We suspect that the severe degradation occurs since the need to split the high-resolution images and more importantly the tendency of PhotoGuard to create very fine perturbations specific to each image make it particularly vulnerable to downsizing. On the other hand, since our model learns perturbations that work well on average across the training data, the perturbations tend to be more coarse. However, creating a more practically robust model requires further investigation and future work to fully reveal these aspects.

Assignment Function. In FastProtect, the assignment function $\mathcal{A}$ groups images as shown in Figure 6c. We observed that each perturbation has characteristic groupings of images. For example, the first perturbation (top left in Figure 6c) predominantly groups images of faces, portraits, or close-up shots of objects, with a notably dark background. The second perturbation (top right) includes images with moderate scene complexity or texture. The third perturbation (bottom left) selects the simplest images in the dataset, particularly cartoons or images with simple objects and minimal backgrounds in the Subject domain. Finally, the fourth perturbation (bottom right) groups the most complex textured images. For instance, in the subject domain, images with detailed textures like grassy backgrounds are mostly selected here, and in the face, cartoon, and painting domains, images with complex and detailed backgrounds or scenes are also grouped in this category.

Adaptive Targeted Protection. Table 6 shows a quantita-

tive report of Figure 7a. Similarly, the performance in each test domain varies according to the pattern repetition of the target image. For example, high repetition target images perform well in the Object domain but poorly in the Face and Cartoon domains. This is because the Object domain mostly contains images with dense textures and complex scenes, which match well with the frequency of high repetition target images. Conversely, the Cartoon domain, which typically has very flat textures, does not benefit from high repetition target images. For the Face domain, even though it is similar to the Object domain in being natural photos, the backgrounds are blurred or monochromatic, and the textures of faces are frequently flat, differing significantly from objects. For the low repetition target images, the opposite trend is observed. Performance decreases in the Object domain but improves in the Face and Cartoon. In the Painting domain, performance remains consistent across all target images. This is because the artworks in the Painting domain exhibit a wide variety of textures and complexities, leading to an average performance across different target images. For instance, the Painting domain dataset includes diverse images ranging from Van Gogh-style oil paintings to black and white sketch drawings.

Comparison at the Same Protection Strength. In the main paper, we compare FastProtect with other protection frameworks by adjusting the protection strength (η) to match the protection level across methods for a fairer comparison (Table 2). The reason for this evaluation setup is that we observed different protection losses produce varying protection-invisibility trade-offs. Therefore, protection efficacy alone cannot be used to judge the superiority of a model. In Figure 10, we report the results by varying the protection strength and analyzing protection efficacy vs. invisibility. However, such benchmark requires significant computational resources, thus we adjust the protection strength to align the trade-off line for comparison.

Still, one might wonder about the performance when the protection strength is fixed for all methods. To answer this possible inquiry, we conduct a comparison with all protection frameworks by fixing the protection strength η at 8, as shown in Table 7. In most cases, FastProtect demonstrates outstanding results in terms of invisibility while achieving moderate protection efficacy. This balance suggests that FastProtect performs exceptionally well when considering the protection-invisibility trade-off. In the Cartoon domain, Impasto [1] shows better results in invisibility, but our method significantly outperforms Impasto in efficacy, which indicates a favorable trade-off trend.

Limitation & Future Directions. As discussed in Section 5, all protection solutions, including our model, still need improvements in terms of invisibility. However, as analyzed by Ahn et al. [1], to prevent mimicry, perturbations must be applied broadly across the image. This characteristic makes it very challenging to achieve a high level of invisibility beyond a certain point.

Moreover, a common phenomenon across all models is that, even when the same budget of perturbation is applied to all images, the degree of protection varies from image to image. Interestingly, if a specific model fails to protect a certain image, other models also tend to struggle with it. This issue likely stems from the fact that all frameworks rely on texture or semantic losses (or both), thus they all share the same vulnerability. A potential solution to this problem is to measure how difficult an image is to protect in advance and dynamically adjust the budget accordingly. However, practically, it is challenging to determine the difficulty of protection before the personalization. Therefore, predicting the protectability of input images and adjusting the intensity of perturbations adaptively is a worth a try research direction. The another advantage of this approach is that easy-to-protect images can be given lower perturbation strength, naturally enhancing invisibility.

E. Additional Results

Figure 11 and 12 show additional qualitative comparisons of existing protection frameworks. In Table 8, 9, 10, and 11, we evaluate the protection frameworks using the additional metrics. In these comparisons, our model demonstrates better invisibility while showing average protection efficacy; similar protection performance with the texture loss model group. Notably, models that primarily utilize texture loss (e.g., PhotoGuard, Mist, SDST, Impasto, and ours; although Mist and SDST utilize both losses, we observed that texture loss has a much higher influence) tend to exhibit relatively high invisibility. Conversely, AdvDM and Anti-DreamBooth, which use semantic loss, show higher protection efficacy in most of the protection efficacy measures. This phenomenon highlights that the different losses exhibit varying trends. Understanding why these losses perform differently across metrics will be crucial for future advancements in protection loss design.

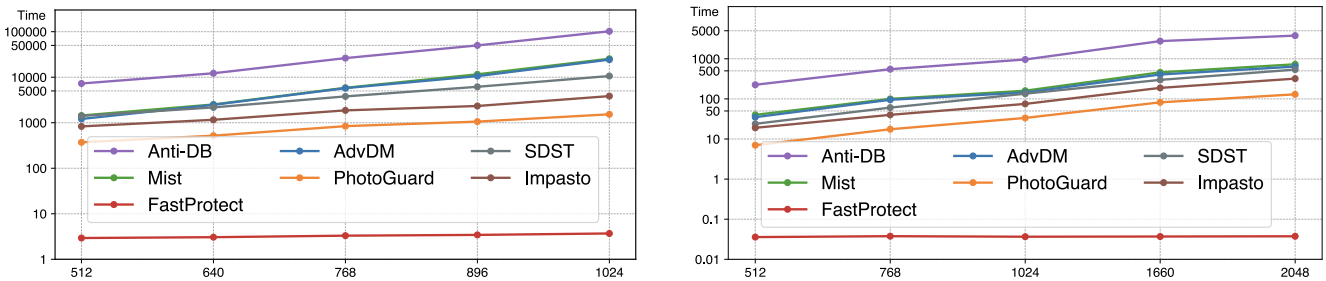
In Figure 9, we compare the latency versus input image resolution on both CPU and GPU. On the CPU (Apple silicon), FastProtect completes the protection in only 4 seconds, while all other methods require significantly more time; 15 minutes for PhotoGuard [46] and 10 hours for Anti-DreamBooth [51]. We also include the protection efficacy versus invisibility trade-off comparison for both the Object and Cartoon domains in Figure 10. FastProtect demonstrates an improved trade-off curve in both domains.

Table 10. Additional quantitative comparison on the Painting domain.

Domain: Painting	Invisibility			Protection Efficacy		
	DISTS (↓)	$\text{LPIPS}_{\text{VGG}}$ (↓)	AHIQ (↑)	FID (↑)	TOPIQ-NR (↓)	QAlign (↓)
AdvDM [28]	0.153	0.267	0.572	**357.6**	**0.438**	2.889
PhotoGuard [46]	**0.107**	0.189	**0.579**	350.9	0.613	3.228
Mist [27]	0.129	0.219	**0.579**	357.0	0.615	3.218
SDST [56]	0.152	0.258	0.583	354.1	0.637	3.288
Anti-DB [51]	0.114	0.184	**0.579**	347.7	0.452	**2.860**
Impasto [1]	0.123	0.221	**0.579**	352.4	0.564	3.141
FastProtect	0.110	**0.180**	0.577	356.1	0.506	2.987

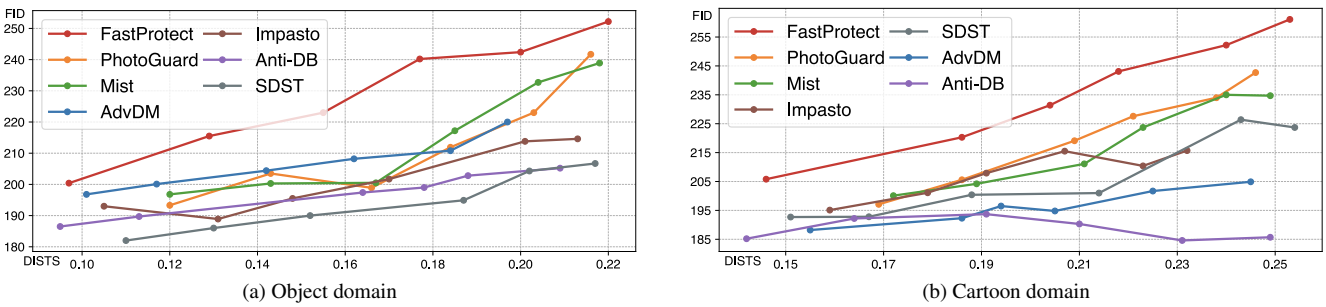
Table 11. Additional quantitative comparison on the Cartoon domain.

Domain: Cartoon	Invisibility			Protection Efficacy		
	DISTS (↓)	$\text{LPIPS}_{\text{VGG}}$ (↓)	AHIQ (↑)	FID (↑)	TOPIQ-NR (↓)	QAlign (↓)
AdvDM [28]	0.271	0.440	0.445	212.5	**0.579**	2.247
PhotoGuard [46]	0.209	0.348	0.587	219.1	0.683	2.618
Mist [27]	0.223	0.365	0.585	223.7	0.681	2.645
SDST [56]	0.237	0.398	0.589	222.7	0.712	2.755
Anti-DB [51]	0.294	0.468	0.445	**225.4**	**0.429**	**2.166**
Impasto [1]	0.207	0.376	**0.602**	215.5	0.590	2.294
FastProtect	**0.186**	**0.301**	0.585	220.3	0.656	2.530



(a) On CPU (Apple M1 Max) (b) On GPU (NVIDIA A100)

Figure 9. Inference latency (log-scaled) vs. image size on both CPU and GPU environments.



(a) Object domain (b) Cartoon domain

Figure 10. Protection efficacy vs. invisibility comparison on the Object and Cartoon domains.

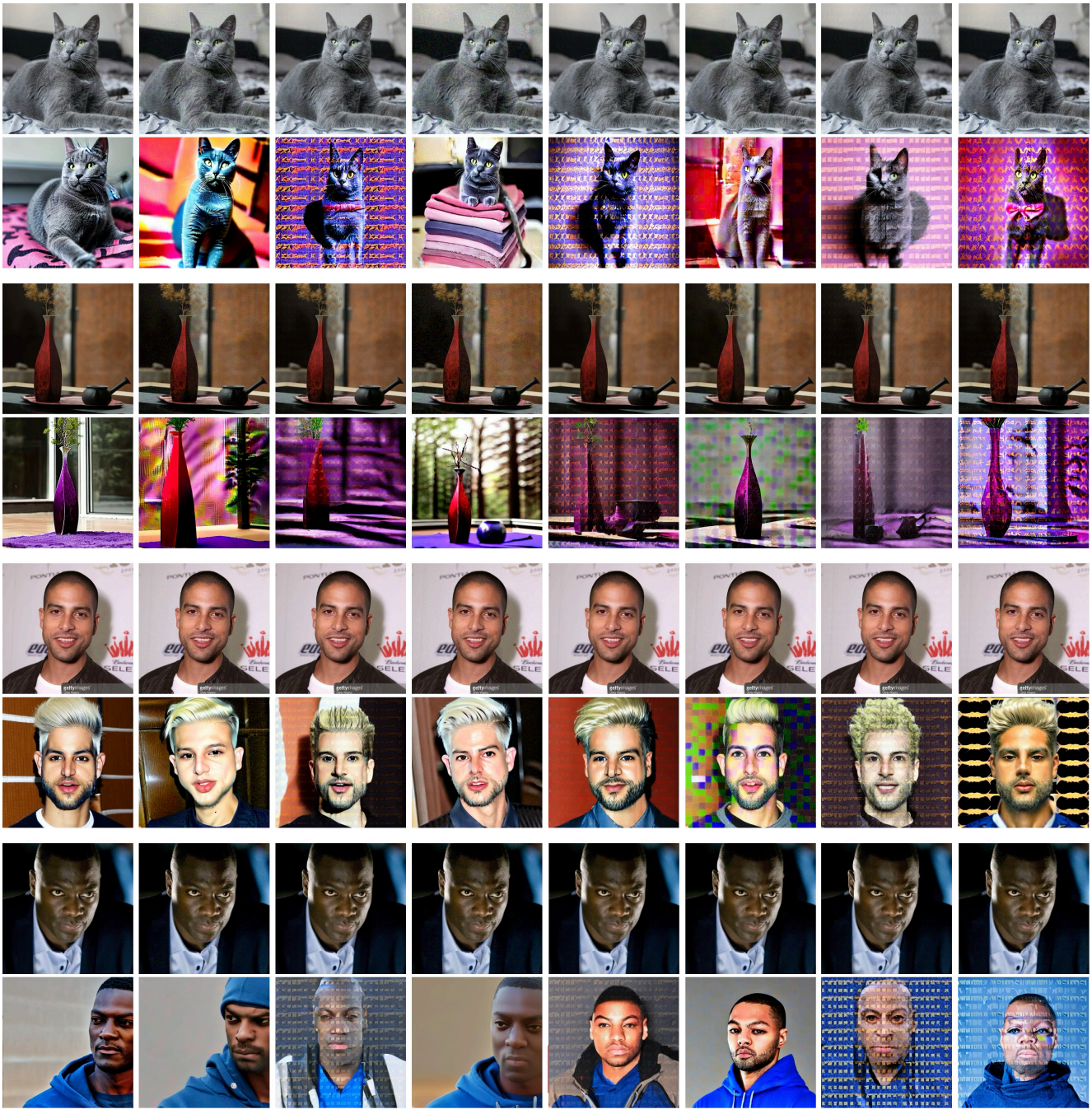


Figure 11. Additional qualitative comparison. For each example, (top) protected image and (bottom) mimicry image generated by LoRA.

Figure 12. Additional qualitative comparison. For each example, (top) protected image and (bottom) mimicry image generated by LoRA.