DiffBreak: Is Diffusion-Based Purification Robust?

Andre Kassis, Urs Hengartner, Yaoliang Yu
Cheriton School of Computer Science, University of Waterloo
Waterloo, Ontario, Canada
{akassis, urs.hengartner, yaoliang.yu}@uwaterloo.ca

Abstract

Diffusion-based purification (*DBP*) has become a cornerstone defense against adversarial examples (*AE*s), regarded as robust due to its use of diffusion models (*DM*s) that project *AE*s onto the natural data manifold. We refute this core claim, theoretically proving that gradient-based attacks effectively target the *DM* rather than the classifier, causing *DBP*'s outputs to align with adversarial distributions. This prompts a reassessment of *DBP*'s robustness, accrediting it two critical factors: inaccurate gradients and improper evaluation protocols that test only a single random purification of the *AE*. We show that when accounting for stochasticity and resubmission risk, *DBP* collapses. To support this, we introduce *DiffBreak*, the first reliable toolkit for differentiation through *DBP*, eliminating gradient mismatches that previously further inflated robustness estimates. We also analyze the current defense scheme used for *DBP* where classification relies on a single purification, pinpointing its inherent invalidity. We provide a statistically grounded majority-vote (*MV*) alternative that aggregates predictions across multiple purified copies, showing partial but meaningful robustness gain. We then propose a novel adaptation of an optimization method against deepfake watermarking, crafting systemic perturbations that defeat *DBP* even under *MV*, challenging *DBP*'s viability.

1 Introduction

ML classifiers are vulnerable to *Adversarial Examples (AEs)*—imperceptible perturbations that induce misclassification [37, 15]. *Adversarial Training* [28, 54] is attack-specific and costly [44], while other defenses [16, 13, 3, 46, 49, 32, 50, 9] are vulnerable to adaptive attacks [38]. *Diffusion-Based Purification (DBP)* [30, 40], which leverages *diffusion models (DMs)* [36, 34], has emerged as a promising defense [6, 45, 7, 53, 31, 42]. *DBP* purifies inputs by solving the reverse stochastic differential equation (*SDE*) associated with *DM*s, diffusing the input with random noise to dilute adversarial changes and iteratively denoising it. Given *DMs'* data modeling abilities [12, 39], *DBP* operates under the assumption that each denoising (reverse) step's output belongs to a corresponding *marginal natural distribution* (a distribution obtained by introducing Gaussian noise into natural inputs, with decreasing variance proportional to that step).

This is the foundation for *DBP*'s *robustness* as it ensures its outputs are from the natural distribution, while *AE*s lie outside this manifold and are unlikely to be preserved. Recent works treat *DBP* as a static pre-processor that standard gradient-based attacks cannot directly manipulate but rather only attempt to evade. Thus, attackers are assumed to either fail due to randomness that masks the classifier's vulnerabilities [26], or be forced to rely on surrogate losses and heuristic approximations [20, 41]. We theoretically refute *DBP*'s robustness rooted in its ability to perform projection onto the natural manifold. This is paradoxical as it assumes correct behavior of the score model s_θ used by the *DM* to mimic the gradients of the marginal distributions. Yet, s_θ is an *ML* system with exploitable imperfections. In §3.1, we prove that **standard** adaptive attacks that simply backpropagate the classifier's loss gradients through *DBP* **effectively target s_θ rather than the classifier**, forcing it to generate samples from an adversarial distribution. Hence, *DBP*'s theory no longer holds.

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

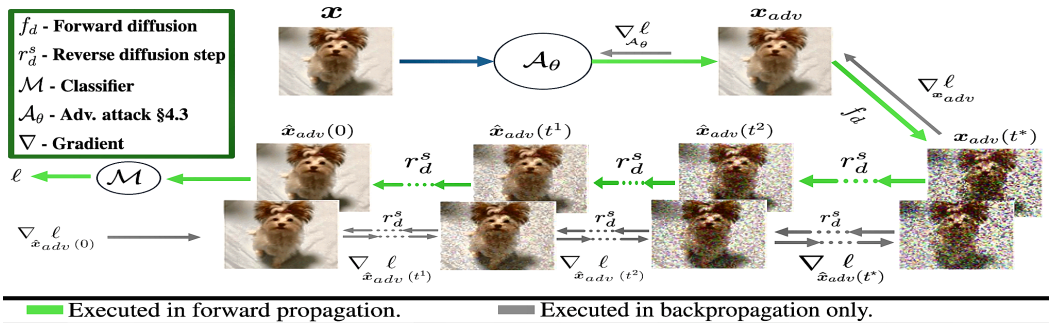


Figure 1: **DiffGrad**. $\boldsymbol{x}$ is given to the iterative attack algorithm $\mathcal{A}_\theta$ to generate $\boldsymbol{x}_{adv}$. At each iteration, $\boldsymbol{x}_{adv}$ is propagated through *DBP*, yielding $\hat{\boldsymbol{x}}_{adv}(0)$ that is given to $\mathcal{M}$ while storing each intermediate $\hat{\boldsymbol{x}}_{adv}(t)$ (bottom replicas) from *DBP*'s reverse pass (see §2) but without saving any graph dependencies. Backpropagation uses the stored samples: Starting from $t=-dt$ ($dt<0$—see §2), each $\hat{\boldsymbol{x}}_{adv}(t)$ is used to recompute $\hat{\boldsymbol{x}}_{adv}(t+dt)$, retrieving the required dependencies. Then, we recursively obtain the gradient w.r.t. $\hat{\boldsymbol{x}}_{adv}(t)$ from the gradients w.r.t. $\hat{\boldsymbol{x}}_{adv}(t+dt)$ using this recovered sub-graph (see §3.2). Finally, gradients are backpropagated in a standard manner from $\boldsymbol{x}_{adv}(t^*)$ to $\mathcal{A}_\theta$ to update $\boldsymbol{x}_{adv}$.

In §3.2 and §3.3, we revisit *DBP*'s previous robustness, attributing it to backpropagation issues and improper evaluation protocols. Most works [20, 30, 26, 40] judge attacks based on a single purification of the final adversarial input. While this seemingly mirrors *DBP*'s intended "purify once and classify" deployment [30, 40], it is statistically invalid: since *DBP* samples noise randomly, one evaluation of the *AE* fails to capture true **single** misclassification probability across possible purifications. Moreover, due to *DBP*'s memory-intensive gradients, prior works implement exact backpropagation [20, 26, 41] via checkpointing [8]. We discover and fix issues in all previous implementations, introducing **DiffGrad**—the first reliable module for exact backpropagation through *DBP* (see Fig.1). In §4, we show that even under the one-sample evaluation protocol, *AutoAttack* [10] significantly outperforms prior works, exposing their gradient fallacies. We further use a multi-sample evaluation that captures *DBP*'s stochasticity and resubmission risk—aligned with **DiffHammer** [41], which also identified the evaluation gap, but retained problematic gradients. By fixing both, we prove *DBP* even more vulnerable than they report, with robustness dropping below 17.19%. Finally, we show prior attack enhancements tailored to *DBP* (e.g., **DiffHammer**, **DiffAttack** [20]), being oblivious to the ability of standard gradient-based attacks to reshape its behavior, in fact, disrupt optimization, lowering attack performance, and confirming our theoretical scrutiny.

To replace the current inherently vulnerable single-evaluation *DBP* defense scheme, we propose a more robust majority-vote (*MV*) setup, where multiple purified copies are classified and the final decision is based on the majority label. Yet, even under *MV*, *DBP* retains only **partial** robustness against norm-bounded attacks—see §4.2, supporting our theoretical findings. As we established the inefficacy of intermediate projections, this resistance is due to *DBP*'s vast stochasticity: common *AE*s that introduce large changes to *individual* pixels are diluted and may not impact most paths. Instead, we require stealthy modifications that affect many pixels. In §5, we propose a novel adaptation of a recent strategy [21] from image watermarking that crafts low-frequency perturbations, accounting for links between neighboring pixels. This technique defeats *DBP* even under *MV*

Contributions. (i) Analytically scrutinizing adaptive attacks on *DBP*, proving they nullify its theoretical robustness claims. (ii) Addressing protocol and gradient issues in prior attacks, enabling reliable evaluations, and demonstrating degraded performance of *DBP* and the ineffectiveness of recent attack enhancement strategies [20, 41]. (iii) Introducing and evaluating a statistically grounded *MV* defense protocol for *DBP*. (iv) Proposing and adapting low-frequency (*LF*) attack optimizations to *DBP*, achieving unprecedented success even under *MV*. (v) Availability: aside from scalability and backpropagation issues, existing *DBP* implementations and attacks lack generalizability. We provide *DiffBreak*[1]—the first toolkit for evaluating any classifier with *DBP* under various optimization methods, including our novel *LF*, using our reliable **DiffGrad** module for backpropagation. (vi) Extensive evaluations on ***ImageNet*** and ***CIFAR-10*** prove our methods defeat *DBP*, bringing its robustness to $\sim 0\%$, outperforming previous works by a large margin.

[1] `https://github.com/andrekassis/DiffBreak`

2 Background & Related Work

Adversarial Attacks. Given $\boldsymbol{x} \in \mathbb{R}^d$ with true label y, classifier $\mathcal{M}$, and preprocessing defense G ($G \equiv Id$ if no defense), attackers aim to generate a *similar* $\boldsymbol{x}_{adv}$ s.t. $\mathcal{M}(G(\boldsymbol{x}_{adv})) \neq y$. Formally:

$$\boldsymbol{x}_{adv} = \underset{\boldsymbol{\mathcal{D}}(\boldsymbol{x}', \boldsymbol{x}) \leqslant \epsilon_{\boldsymbol{\mathcal{D}}}}{\arg\min} \mathbb{E}[\ell_G^{\mathcal{M}}(\boldsymbol{x}',\ y)]$$

for loss $\ell_G^{\mathcal{M}}$. Typically, $\ell_G^{\mathcal{M}}(\boldsymbol{x}, y) = \ell(\mathcal{M}(G(\boldsymbol{x})), y)$, where ℓ is a loss over $\mathcal{M}$'s output that captures the desired outcome. For instance, $\ell(\mathcal{M}(G(\boldsymbol{x})), y)$ can be chosen as the probability that the classifier's output label is y, which we strive to minimize. $\boldsymbol{\mathcal{D}}$ is a distance metric that ensures similarity if kept below some $\epsilon_{\boldsymbol{\mathcal{D}}}$. These *untargeted* attacks are the focus of many works [30, 40, 32, 29]. The expected value accounts for potential stochasticity in G (e.g., *DBP* below).

Diffusion models (DMs) [34, 36] learn to model a distribution p on $\mathbb{R}^d$ by reversing the process that diffuses inputs into noise. *DM*s involve two stochastic processes. The forward pass converts samples into pure Gaussians, and is governed by the following *SDE* for an infinitesimal step $dt > 0$:

$$d\boldsymbol{x} = \boldsymbol{f}(\boldsymbol{x},\ t)dt + g(t)d\boldsymbol{w}. \tag{1}$$

eq. (1) describes a stochastic integral whose solution up to $t^* \in [0, 1]$ gives $\boldsymbol{x}(t^*)$. Here, $\boldsymbol{f} : \mathbb{R}^d \times \mathbb{R} \longrightarrow \mathbb{R}^d$ is the drift, $\boldsymbol{w} : \mathbb{R} \longrightarrow \mathbb{R}^d$ is a Wiener process, and $g : \mathbb{R} \longrightarrow \mathbb{R}$ is the diffusion coefficient. We focus on **VP-SDE**, which is the most common *DM* for *DBP* [30, 48, 40]. Yet, our insights generalize to all *DM*s (see [36] for a review). In **VP-SDE**, $\boldsymbol{f}(\boldsymbol{x},\ t) = -\frac{1}{2}\beta(t)\boldsymbol{x}(t)$ and $g(t) = \sqrt{\beta(t)}$, where $\beta(t)$ is a noise scheduler outputting small positive constants. These choices yield the solution:

$$\boldsymbol{x}(t^*) = \sqrt{\alpha(t^*)}\boldsymbol{x} + \sqrt{1 - \alpha(t^*)}\boldsymbol{\epsilon} \tag{2}$$

for $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}_d)$ and $\alpha(t) = e^{-\int_0^t \beta(s)\, ds}$. With proper parameters, we have $\boldsymbol{x}(1) \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}_d)$. Thus, a process that inverts *eq. (1)* from $t^* = 1$ to 0 allows generating samples in p from random noise. Due to Anderson [1], the reverse pass is known to be a stochastic process with:

$$d\hat{\boldsymbol{x}} = [\boldsymbol{f}(\hat{\boldsymbol{x}}(t),\ t) - g^2(t)\nabla_{\hat{\boldsymbol{x}}(t)} \log p_t(\hat{\boldsymbol{x}}(t))]dt + g(t)d\bar{\boldsymbol{w}}. \tag{3}$$

Defining $\hat{\boldsymbol{x}}(t^*) := \boldsymbol{x}(t^*)$, the process evolves from t^* to 0 with a **negative** time step dt and reverse-time Wiener process $\bar{\boldsymbol{w}}(t)$. Let $p(\boldsymbol{x})$ be the probability of $\boldsymbol{x}$ under p, and $p_{0t}(\tilde{\boldsymbol{x}}|\boldsymbol{x})$ the conditional density at t given $\boldsymbol{x}(0) = \boldsymbol{x}$. Then, the marginal density is given by $p_t(\tilde{\boldsymbol{x}}) = \int p(\boldsymbol{x}) p_{0t}(\tilde{\boldsymbol{x}}|\boldsymbol{x}) d\boldsymbol{x}$, where $p_0 \equiv p$. Solving *eq. (3)* requires the **score** $\nabla_{\hat{\boldsymbol{x}}(t)} \log p_t(\hat{\boldsymbol{x}}(t))$, which can be approximated via a trained model $\boldsymbol{s}_\theta$ s.t. $\boldsymbol{s}_\theta(\hat{\boldsymbol{x}}(t), t) \approx \nabla_{\hat{\boldsymbol{x}}(t)} \log p_t(\hat{\boldsymbol{x}}(t))$ [36], yielding:

$$d\hat{\boldsymbol{x}} = -\frac{1}{2}\beta(t)[\hat{\boldsymbol{x}}(t) + 2\boldsymbol{s}_\theta(\hat{\boldsymbol{x}}(t), t)]dt + \sqrt{\beta(t)}d\bar{\boldsymbol{w}}. \tag{4}$$

As no closed-form solution exists, the process runs iteratively over **discrete** negative steps dt. Starting from t^*, $d\hat{\boldsymbol{x}}$ is calculated at each $i = |\frac{t}{dt}|$, until $t = 0$. This *continuous-time DM* describes a stochastic integral (despite the discretized implementations). An alternative, *Denoising diffusion probabilistic modeling* (**DDPM**) [19, 34], considers a *discrete-time DM*. The two are equivalent (see Appendix B).

DBP [30, 48, 40, 51] performs purification by diffusing each input $\boldsymbol{x}$ until optimal time t^* that preserves class semantics while still diminishing malicious interruptions. $\boldsymbol{x}(t^*)$ is then given to the reverse pass, reconstructing a clean $\hat{\boldsymbol{x}}(0) \approx \boldsymbol{x}$ s.t. $\hat{\boldsymbol{x}}(0) \sim p$ for correct classification.

Certified vs. Empirical Robustness. While recent work has explored certified guarantees for *DBP* via randomized smoothing (RS) [48, 5, 55, 6, 45], these guarantees hold only under restrictive assumptions — e.g., small ℓ_2 perturbations and thousands of Monte Carlo samples per input. In contrast, empirical defenses [40, 30, 31, 7, 53] aim for practical robustness under realistic threats [10] and efficient inference. Like prior work [41, 20, 30, 40, 24], we explicitly target these empirical defenses—not orthogonal certified variants whose guarantees do not hold under stronger practical perturbations or operational constraints. Certification protocols are computationally prohibitive, and their bounds fail to capture threats that remain imperceptible but exceed certified radii.

3 Revisiting Diffusion-Based Purification

We study gradient-based attacks. In §3.1, we theoretically prove that adaptive attacks invalidate *DBP*'s principles. Then, we reconcile this with previous findings of *DBP*'s robustness, attributing them to

backpropagation issues and improper evaluation protocols. In §3.2, we analyze backpropagation mismatches in previous works and propose fixes. Finally, we present an improved evaluation protocol for *DBP* in §3.3 to better measure its robustness.

3.1 Why *DBP* Fails: Theoretical Vulnerability to Adaptive Attacks

DBP's robustness is primarily attributed to its ability to project the purified $\hat{\boldsymbol{x}}(0)$ onto the natural manifold [30, 48, 20]. This foundational assumption—rooted in the inherent behavior of the purification process—has remained unchallenged. In fact, the most advanced attacks have been explicitly tailored to exploit or circumvent it: **DiffAttack** [20] introduces suboptimal per-step losses to perturb the purification trajectory (see §4), while **DiffHammer** [41] constrains its optimization to feasible paths that evade detection by the *DBP* process itself. *DBP* is often justified through its marginal consistency: since $\boldsymbol{s}_\theta \approx \nabla \log p_t$, $\{\boldsymbol{x}(t)\}_{t\in[0,1]}$ and $\{\hat{\boldsymbol{x}}(t)\}_{t\in[0,1]}$ follow the same marginals [1], yielding $\hat{\boldsymbol{x}}(0) \sim p$. Specifically, Xiao et al. [48] show that:

$$\Pr(\hat{\boldsymbol{x}}(0)|\boldsymbol{x}) \propto p(\hat{\boldsymbol{x}}(0)) \cdot e^{-\frac{\alpha(t^*)\|\hat{\boldsymbol{x}}(0)-\boldsymbol{x}\|_2^2}{2(1-\alpha(t^*))}}$$

where p is the density of the natural data distribution and $\alpha(t^*)$ is the variance schedule at time t^*. Thus, *DBP* is expected to reject adversarial examples by construction: the probability of producing any $\hat{\boldsymbol{x}}(0)$ that is both adversarial and lies off-manifold is exponentially suppressed by the score model. However, this reasoning assumes the reverse process remains faithful to p. In practice, the score function $\boldsymbol{s}_\theta$ is itself an ML model—differentiable and susceptible to adversarial manipulation. Our key insight is that traditional gradient-based attacks, when backpropagated through *DBP*, do not simply bypass the purifier—they implicitly target it, steering the score model to generate samples from an adversarial distribution. This challenges the assumptions underlying prior attack strategies, many of which distort gradients or constrain optimization in ways that ignore the score model's vulnerability, undermining their own effectiveness. Below, we formally characterize this vulnerability.

Definitions. *Diffusion Process & Score Model.* For $t_1 \geqslant t_2$, let $\hat{\boldsymbol{x}}_{t_1:t_2}$ represent the joint reverse diffusion trajectory $\hat{\boldsymbol{x}}(t_1), \hat{\boldsymbol{x}}(t_1+dt), \ldots, \hat{\boldsymbol{x}}(t_2)$. Let $\boldsymbol{s}_\theta$ denote the score model used by *DBP* to approximate the gradients of the natural data distribution at different time steps. $\boldsymbol{s}_{\theta^t}$ is the abstract score model invoked at time step t with parameters θ^t, corresponding to $\boldsymbol{s}_\theta(\cdot, t)$. Given that the score model $\boldsymbol{s}_\theta$ is an ML model that interacts with adversarial input $\boldsymbol{x}$, we denote the parameters at each reverse step as $\theta^t_{\boldsymbol{x}}$, capturing their dependence on $\boldsymbol{x}$. That is, $\theta^t_{\boldsymbol{x}} \equiv \theta^t(\boldsymbol{x})$, which makes explicit that the purification process is **not immutable**, as adversarial modifications to $\boldsymbol{x}$ can shape its behavior.

Classifier. For a classifier $\mathcal{M}$ and label y, let $\mathcal{M}^y(\boldsymbol{u})$ denote the probability that $\boldsymbol{u}$ belongs to class y.

Adaptive Attack. A gradient-based algorithm that iteratively updates $\boldsymbol{x}$ with learning rate $\eta > 0$:

$$\boldsymbol{x} = \boldsymbol{x} - \eta\widetilde{\nabla}_{\boldsymbol{x}}, \quad \text{where} \quad \widetilde{\nabla}_{\boldsymbol{x}} = \frac{1}{N}\sum_{n=1}^{N} \nabla_{\boldsymbol{x}}[\mathcal{M}^y(\hat{\boldsymbol{x}}(0)_n)].$$

Here, $\widetilde{\nabla}_{\boldsymbol{x}}$ is the empirical gradient estimate over N purified samples $\hat{\boldsymbol{x}}(0)_n$, obtained via Monte Carlo approximation and backpropagated through *DBP*'s stochastic process.

Theorem 3.1. *The adaptive attack **optimizes the entire reverse diffusion process**, modifying the parameters $\{\theta^t_{\boldsymbol{x}}\}_{t\leqslant t*}$ such that the output distribution $\hat{\boldsymbol{x}}(0) \sim DBP^{\{\theta^t_{\boldsymbol{x}}\}}(\boldsymbol{x})$, where $DBP^{\{\theta^t_{\boldsymbol{x}}\}}(\boldsymbol{x})$ is the DBP pipeline with the score model's weights $\{\theta^t_{\boldsymbol{x}}\}_{t\leqslant t*}$ adversarially aligned. That is, the adversary **implicitly controls the purification path** and optimizes the weights $\{\theta^t_{\boldsymbol{x}}\}_{t\leqslant t*}$ to maximize:*

$$\max_{\{\theta^t_{\boldsymbol{x}}\}_{t\leqslant t*}} \mathbb{E}_{\hat{\boldsymbol{x}}(0)\sim DBP^{\{\theta^t_{\boldsymbol{x}}\}}(\boldsymbol{x})}[\Pr(\neg y|\hat{\boldsymbol{x}}(0))].$$

*Since $\{\theta^t_{\boldsymbol{x}}\}_{t\leqslant t*}$ depend on the purification trajectory $\hat{\boldsymbol{x}}_{t*:0}$, optimizing $\boldsymbol{x}$ under the adaptive attack **directly shapes the purification process itself**, forcing DBP to generate adversarially structured samples that maximize misclassification.*

Impact. As *DBP* is widely perceived as immutable, its robustness is often attributed to gradient masking (vanishing or explosion) due to its presumed ability to project inputs onto the natural manifold. However, this reasoning treats *DBP* as a **static pre-processing step** rather than a **dynamic optimization target**. Our result **overturns this assumption**: rather than resisting adversarial influence, *DBP* itself becomes an **active participant in adversarial generation**. Crucially, standard

adaptive attacks that backpropagate accurate gradients through *DBP* do not merely evade it—they implicitly **exploit and reshape** its behavior, forcing the purification process to align with the attacker's objective. The introduced perturbations **directly shape the purification trajectory**, causing *DBP* to generate, rather than suppress, adversarial samples. Thus, *DBP* **does not inherently neutralize adversarial perturbations** but instead shifts the optimization target from the classifier to the score model s_θ, leaving this paradigm highly vulnerable.

Proof. (Sketch) The adaptive attack maximizes the probability that the classifier mislabels the purified output, which is expressed as an expectation over the stochastic purification process. Applying the law of the unconscious statistician (LOTUS), we reformulate this expectation over the joint distribution of the reverse diffusion trajectory, shifting the dependency away from the classifier's decision at the final step. Leveraging smoothness, we interchange differentiation and expectation, revealing that the adaptive attack's gradient corresponds to the expected gradient of the purification trajectory, which is governed by the score model's parameters. Crucially, these parameters are optimized implicitly through perturbations to $\boldsymbol{x}$, rather than being explicitly modified. This demonstrates that the adaptive attack does not merely navigate *DBP*—it fundamentally shapes its behavior, exploiting the score model rather than directly targeting the classifier. The full proof is in Appendix C. □

3.2 Precise *DBP* Gradients with DiffGrad

Building on our theoretical analysis, we examine why prior adaptive attacks [30, 40, 20, 26, 24] underperform and show how **DiffGrad**, provided by our *DiffBreak* toolkit, resolves these limitations.

We reserve the terms **forward/reverse pass** for *DBP* and **forward/backpropagation** for gradient computations. Let $\hat{\boldsymbol{x}}(t^*) := \boldsymbol{x}(t^*)$. The reverse process follows:

$$\hat{\boldsymbol{x}}(t + dt) = \hat{\boldsymbol{x}}(t) + d\hat{\boldsymbol{x}}(t), \quad dt < 0, \tag{5}$$

and by chain rule the gradient of any function F of $\hat{\boldsymbol{x}}(t + dt)$ w.r.t. $\hat{\boldsymbol{x}}(t)$ is:

$$\nabla_{\hat{\boldsymbol{x}}(t)} F = \nabla_{\hat{\boldsymbol{x}}(t)} \langle \hat{\boldsymbol{x}}(t + dt), \nabla_{\hat{\boldsymbol{x}}(t+dt)} F \rangle. \tag{6}$$

Applying this recursively yields gradients w.r.t. $\hat{\boldsymbol{x}}(t^*)$, and via eq. (2), the input $\boldsymbol{x}$. Yet, standard automatic differentiation is impractical for *DBP* due to excessive memory overhead from dependencies between all $\hat{\boldsymbol{x}}(t)$. Prior work resorts to approximations such as the **adjoint** method [25], or checkpointing [8] to compute the exact gradients. However, approximations yield suboptimal attack performance [24]. On the other hand, we identify critical issues in existing checkpointing-based implementations that previously led to inflated estimates of *DBP*'s robustness (see Appendix D for detailed analysis, **DiffGrad**'s pseudo-code, and empirical evaluations of backpropagation issues):

1) High-Variance Gradients: Gradient-based attacks require estimating expected gradients using N Monte Carlo (EOT) samples. When N is small, gradient variance is high, leading to unreliable updates. Prior works used small N due to limitations in how EOT samples were purified—one at a time via serial loops in standard attack benchmarks (e.g., *AutoAttack* [10]). **DiffGrad** parallelizes EOT purifications. This integrates seamlessly with standard attacks. Coupled with our termination upon success protocols—see §4.2, this enables us to use up to 128 EOT samples (for ***CIFAR-10***) with a speedup of up to $41\times$ upon early termination, drastically reducing variance.

2) Time Consistency: torchsde[2] is the de facto standard library for *SDE* solvers. Hence, several prior checkpointing approaches likely use *torchsde* as their backend. Yet, *torchsde* internally converts the integration interval into a *PyTorch* tensor, which causes a discrepancy in the time steps on which the score model is invoked during both propagation phases due to rounding issues if the checkpointing module is oblivious to this detail. **DiffGrad** ensures time steps match in both phases.

3) Reproducing Stochasticity: Forward and backward propagation should reuse identical randomness to preserve fidelity. In vanilla *DBP*, diffusion noise cancels in the gradient, so not preserving stochastic components has little effect. Yet, for guided schemes whose computations depend on randomness (some current ones and potential future variants), failing to reproduce the noise realizations can bias gradients and derail optimization. We thus introduce a structured *Noise Sampler* $\boldsymbol{NS}$ that records all noise during the forward pass and reuses it in backpropagation. We also enable deterministic CuDNN kernel selection to avoid backend nondeterminism—see Appendix D.1.3 for details.

[2] https://github.com/google-research/torchsde

4) Guidance Gradients: Guided *DBP* [40] uses guidance metrics along the reverse pass. This guidance is obtained by applying a guidance function $\boldsymbol{g_{fn}}$ to (potentially) both the original input $\boldsymbol{x}$ and the reverse-diffused $\hat{\boldsymbol{x}}(t)$. As such, it creates paths from $\boldsymbol{x}$ to the loss that basic checkpointing fails to consider. Furthermore, the guidance itself is often in the form of a gradient, necessitating second-order differentiation during backpropagation, which is, by default, disabled by differentiation engines. We extend **DiffGrad** to include these "guidance" gradients. Critically, recent SOTA *DBP* defenses generate purified outputs from pure noise, relying on $\boldsymbol{x}$ only via guidance (see §6). As such, the lack of support for guidance gradients in all previous implementations leads to a false sense of security. To our knowledge, **DiffGrad** is the first to account for this component.

5) Proper Use of The Surrogate Method [24]: Upon inspecting **DiffHammer**'s code, we find their checkpointing implementation effectively applies Lee and Kim [24]'s **surrogate** approximation to calculate the gradients, which slightly degrades gradient quality with no practical performance gains—see Appendix F. Yet, the real issue lies in **DiffHammer**'s implementation of the **surrogate**, which diverges from [24], further corrupting the gradients. We defer details to Appendix D.1.5.

3.3 On The Issues in *DBP*'s Evaluation Protocols.

DBP is typically deployed in a single-purification (*SP*) setting: an input $\boldsymbol{x}$ is purified once via a stochastic reverse process and classified as $c = \mathcal{M}(\hat{\boldsymbol{x}}(0))$ [30, 40]. This assumes $\hat{\boldsymbol{x}}(0)$ lies on the natural manifold, yielding a correct label with high probability—see §3.1. Yet, prior studies [20, 30, 40] fail to assess this very property: they evaluate robustness by purifying only the final adversarial iterate, testing just a single stochastic path, yielding noisy, unrepresentative estimates.

DiffHammer [41] addressed this with worst-case robustness (Wor.Rob): the attack succeeds if any of N purifications (evaluated at each attack step) leads to misclassification. The metric is Wor.Rob $:= 1 - \frac{1}{S}\sum_{j=1}^{S} \underset{i\in[\![N]\!]}{max}\mathcal{A}_i^{(j)}$, where $\mathcal{A}_i^{(j)} = 1$ if the i-th purification of sample j fails, and S is the dataset size. They justify this protocol by modeling resubmission attacks (e.g., in login attempts), thereby limiting N to (typically) $N = 10$ after which the attacker will be blocked. However, attackers can retry inputs arbitrarily in stateless settings like spam, CSAM, and phishing. Even in stateful systems, the defender has no control over the stochastic path. Thus, *SP* evaluations require many purifications per input for statistically meaningful conclusions.

Yet, any stochastic defense fails given enough queries [27]. Even if *DBP* nearly always projects to the manifold, *SP* is only reliable if misclassification probability is negligible over all paths—rare in practice (§4). Robust predictions must hence aggregate over multiple purifications. We thus propose:

Majority Vote (MV). Given input $\boldsymbol{x}$, generate K purifications and predict by majority: $c = MV^K(\boldsymbol{x}) := mode\{\mathcal{M}(\hat{\boldsymbol{x}}^{(1)}(0)), \ldots, \mathcal{M}(\hat{\boldsymbol{x}}^{(K)}(0))\}$. Unlike certified smoothing [48], *MV* requires no excessive sampling, yet offers a stable, variance-tolerant robustness estimate, albeit without formal guarantees. We use the majority-robustness metric: MV.Rob $:= 1 - \frac{1}{S}\sum_{j=1}^{S} \mathbb{1}\left[MV^K(\boldsymbol{x}_j) \neq y_j\right]$, where y_j is the ground-truth label. **DiffHammer**'s additional Avg.Rob $:= 1 - \frac{1}{NS}\sum_{j=1}^{S}\sum_{i=1}^{N}\mathcal{A}_i^{(j)}$ averages path-level failures across all samples, conflating individual copy failures with sample robustness. A few fragile samples can dominate this metric, making it unreliable for deployment.

4 Experiments

We reevaluate *DBP*, demonstrating its degraded performance when evaluation and backpropagation issues are addressed, cementing our theoretical findings from §3.1.

Setup. We evaluate on ***CIFAR-10*** [22] and ***ImageNet*** [11] similar to previous work [41, 20, 30, 40]. We consider two foundational *DBP* defenses: The **VP-SDE** *DBP* (*DiffPure*) [30] and the *Guided*-**DDPM** (see §3.2), *GDMP* [40]. We use the *DM*s [12, 19, 36] studied in the original works, adopting the same purification settings—See Appendix E. Following Nie et al. [30], we use WideResNet-28-10 and WideResNet-70-16 [52] for ***CIFAR-10***, and WideResNet-50-2 and DeiT-S [14] for ***ImageNet***. As in previous work [40, 30, 20, 41, 26, 24], we focus on the white-box setting and use *AutoAttack*-ℓ_∞ (*AA*–ℓ_∞) [10] with $\epsilon_\infty = 8/255$ for ***CIFAR-10*** and $\epsilon_\infty = 4/255$ for ***ImageNet***. Existing works focus on norm-bounded (ℓ_∞ and ℓ_2). For *DBP*, ℓ_∞ has repeatedly proven superior [20, 24, 26], making it the focus of our evaluations. We report *clean accuracy* (*Cl-Acc*)—without attacks— and *robust accuracy*

(*Rob-Acc*)—the fraction of correctly classified attack samples. We use 256 random test samples per dataset, consistent with prior *DBP* work [41, 7]. Because *DBP* is a compute-intensive defense, we prioritize **breadth over scale**—two datasets (***CIFAR-10***, ***ImageNet***), multiple classifiers and *DBP* schemes, diverse attacks and baselines, and *MV* ablations—so readers see the same qualitative conclusions across settings (see §4- §6; extended results in Appendix F-Appendix H).

4.1 Reassessing One-Shot *DBP* Robustness with Accurate Gradients

As noted in §3, *DBP*'s robustness stems from two factors: inaccurate gradients and improper evaluation. As our work offers enhancements on both fronts, we evaluate each factor separately. Here, we isolate the gradient issue by re-running prior experiments under the same (problematic) single-evaluation protocol, but with accurate gradients via our **DiffGrad** module and compare the results to those from the literature. Despite using only 10 optimization steps (vs. up to 100 in prior work), our method significantly outperforms all gradient approximations and even recent full-gradient methods. For *DiffPure*, we lower robust accuracy from 62.11% (reported by Liu et al. [26]) to 48.05%, and for *GDMP*, from 24.53% (Surrogate [24]) to 19.53%. These results expose the issues in existing gradient implementations and invalidate claimed gains from enhancement strategies like **DiffAttack**. Full breakdowns and additional baselines are provided in Appendix F.

Evaluations Under Liu et al. [26]'s Protocol. Liu et al. [26], like **DiffHammer** [41] and our own analysis, note that evaluating a single purification at attack termination inflates robustness scores. To address this, they propose a refined 1-evaluation protocol, which tests 20 purifications of the final *AE* and declares success if any fail. This offers a stricter assessment than earlier one-shot methods, though still weaker than Wor.Rob (see §3.3). Accordingly, Liu et al. [26] group their method with one-shot evaluations. Repeating their setup using **DiffGrad** (20 attack iterations, $N = 10$ EOT), we observe a dramatic drop in robust accuracy: on WideResNet-28-10, we improve upon Liu et al. [26]'s results by 25.39% and 30.42% for *DiffPure* and *GDMP*, respectively—bringing the Rob-Accs down to 30.86% and 10.55%. These results confirm the superiority of **DiffGrad**'s gradients and expose *DBP*'s realistic vulnerability. Detailed results are in Appendix G.

4.2 *DBP* Under Realistic Protocols

Here, we continue to highlight the shortcomings of one-shot evaluation and the strength of **DiffGrad**, while exposing the invalidity of previous attack enhancements over the standard gradient-based methodologies. To do so, we adopt a Wor.Rob protocol, similar to **DiffHammer**—see §3.3 and also include our majority-vote variant (MV.Rob). We reimplement **DiffHammer** and **DiffAttack** using **DiffGrad**, and compare them to their original versions from [41] and our standard *AA*-ℓ_∞. All evaluations use ***CIFAR-10*** with WideResNet-70-16; We also report ***ImageNet*** results below. **Full** denotes the standard AA attack that uses the full exact gradients (i.e., via checkpointing).

Table 1: *AA*-ℓ_∞ performance comparison on ***CIFAR-10*** (ϵ_∞=$8/255$) under realistic threat models. Metrics include Wor.Rob and MV.Rob under a 10-evaluation protocol. † indicates strategy is *PGD*.

Pur.	Gradient Method	Wor.Rob % Cl-Acc	Wor.Rob % Rob-Acc	MV.Rob % Cl-Acc	MV.Rob % Rob-Acc
DiffPure [30]	**BPDA**	89.06	32.81	91.02	72.27
	DiffAttack (DiffHammer [41])		33.79		NA
	DiffAttack-DiffGrad		15.63		39.45
	DiffHammer (DiffHammer [41])		22.66		NA
	DiffHammer-DiffGrad		10.16		38.28
	Full (DiffHammer [41]) †		36.91		NA
	Full-DiffGrad		**17.19**		**39.45**
GDMP [40]	**BPDA**	91.80	27.73	92.19	53.52
	DiffAttack (DiffHammer [41])		37.7		NA
	DiffAttack-DiffGrad		3.91		14.45
	DiffHammer (DiffHammer [41])		27.54		NA
	DiffHammer-DiffGrad		7.81		25.39
	Full (DiffHammer [41]) †		31.05		NA
	Full-DiffGrad		**7.03**		**16.8**

Choice of batch size (N): We set $N = 10$ for ***CIFAR-10*** to match [41] (which uses $N = 10$ for both EOT and evaluation copies) and to balance robustness/gradient accuracy with runtime and memory; these values reflect practical deployment and evaluation constraints. For ***ImageNet***, we choose $N = 8$ for similar reasons. Ablations with larger N, latency measurements, and practical recommendations appear in Appendix H; complexity details are in Appendix D.1.3 and Appendix D.3.

Results. Table 1 reports Wor.Rob and MV.Rob scores. The attacks run for 100 iterations, terminating upon success. For all attacks—**Full**, **DiffAttack**, and **DiffHammer**—**DiffGrad** yields significantly lower Wor.Rob compared to [41], confirming the gradient mismatches discussed in §3.2 (MV.Rob is unique to our work). This establishes the need for our reliable **DiffGrad** as an essential tool for

future progress in the field, given the repeated problems that continue to surface in implementations of the checkpointing method. With correct gradients (i.e., **DiffGrad**), all three attacks yield Wor.Rob $< 20\%$; for *GDMP*, $< 10\%$, exposing the failure of the single-purification defense and reinforcing the claim that a statistically resistant alternative (e.g., our MV.Rob) must be used with *DBP*. Similarly, this highlights the issues in the attack evaluation protocols that consider a single sample (i.e., 1-evaluation) as they drastically inflate robustness estimates.

Notably, minor Wor.Rob differences among attacks in this range where predictions are highly noisy reflect random variation, not real gains. Hence, we must focus on the MV.Rob when comparing the three: For the same N, our *MV* is strictly harder as it takes the majority label meaning success requires misclassifying $> \lfloor N/2 \rfloor$ of the N purified copies, whereas Wor.Rob counts a single error. Thus, MV.Rob is an upper bound on Wor.Rob, which aligns with the empirical findings attained with **DiffGrad**, showing significant robustness gains. Violations arise only when our MV.Rob results are compared to the Wor.Rob values reported by **DiffHammer** [41] and reflect implementation differences in gradient computation. As **DiffGrad** removes backpropagation mismatches that can reduce gradient fidelity and inflate robustness (both Wor.Rob and MV.Rob), *MV*-versus-worst-case comparisons should be made using a common, **DiffGrad** implementation of each attack.

Throughout §4, we have thus far shown that *DBP*'s reported robustness has been overstated due to gradient inaccuracies. Yet, some recent works introduce *DBP*-specific attack augmentations to improve success rates, but these overlook our theoretical finding that accurate gradients alone can reshape *DBP*'s output distribution—rendering such enhancements unnecessary. **DiffHammer** builds on the assumption that *DBP*'s trajectories split into "vulnerable" and "non-vulnerable" groups, with misleading gradients from the latter, but our analysis (see **Theorem 3.1**) shows that correct differentiation allows the attack to steer *DBP*'s stochastic process toward adversarial outcomes, contradicting that fixed partition. Likewise, **DiffAttack**'s per-step losses artificially strive to alter the output distribution, ignoring the standard gradient-based attack's inherent ability to do so, which could lead to suboptimal optimization updates. Below, we show that both these methods offer no advantages over the standard (full-gradient) attack and can, in fact, be counter-productive.

Under MV.Rob, **DiffAttack-DiffGrad** and **Full-DiffGrad** are identical on *DiffPure*, and **DiffHammer-DiffGrad** leads to MV.Rob lower by a mere 1.17%, which amounts to only 3 samples out of 256 and is thus statistically insignificant. On *GDMP*, **DiffHammer-DiffGrad** performs significantly worse than both others. Hence, **DiffHammer** worsens attack performance in accordance with our theory. **DiffAttack-DiffGrad** matches **Full-DiffGrad** on *DiffPure*; on *GDMP*, it shows a 2.35% edge, which despite the questionable statistical significance given the test set size, could indicate a potential advantage. Yet, our ***ImageNet*** comparisons below refute this hypothesis.

Table 2: *AA*-ℓ_∞ comparison on ***ImageNet*** ($\epsilon_\infty = 4/255$).

Models	Pur.	Gradient Method	Wor.Rob % Cl-Acc	Wor.Rob % Rob-Acc	MV.Rob % Cl-Acc	MV.Rob % Rob-Acc
WideResNet-50-2	*DiffPure* [30]	**Full-DiffGrad**	74.22	**12.11**	77.02	**29.69**
DeiT-S	*DiffPure* [30]	**DiffAttack-DiffGrad**	73.63	25	77.34	42.21
		Full-DiffGrad		**21.09**		**32.81**
	GDMP [40]	**Full-DiffGrad**	69.14	**20.70**	75.0	**32.83**

ImageNet. We evaluate AA-ℓ_∞ on ***ImageNet*** using WideResNet-50-2 and DeiT-S classifiers under $\epsilon_\infty = 4/255$ (100 iterations), following standard practice. For DeiT-S, we also reimplement **DiffAttack** via **DiffGrad** for comparison. We use 16 EOT samples (two batches of $N = 8$) and 8 samples for prediction (Wor.Rob/MV.Rob). As with ***CIFAR-10***, robustness drops sharply: Wor.Rob ranges from just 12.11% to 21.09%, while MV.Rob peaks at 32.83%. This confirms the vulnerability of single-purification and the strength of gradient-based attacks. **DiffAttack** underperforms our standard attack by 9.4% MV.Rob on DeiT-S, reinforcing its inferiority.

5 Defeating Increased Stochasticity

DBP's stochasticity boosts its robustness under *MV* (see §4.2). Typical adversarial strategies incur high-frequency changes as they directly operate on pixels, altering each significantly w.r.t. its neighbors. This leads to visual inconsistencies, limiting the distortion budget. Such modifications are also easily masked by *DBP*'s noise. Instead, systemic, low-frequency (*LF*) changes allow larger perturbations and resist randomness.

Our *LF* method is inspired by a recent attack—***UnMarker*** [21]—on image watermarking that employs novel units termed *Optimizable Filters (OFs)*. In signal processing, a filter is a smoothing operation

defined by a kernel $\mathcal{K} \in \mathbb{R}_{+}^{M \times N}$ (with values that sum to 1), with which the input is convolved. The output at each pixel is a weighted average of all pixels in its $M \times N$ vicinity, depending on the weights assigned by $\mathcal{K}$. Hence, filters incur systemic changes. Yet, they apply the same operation universally, unable to produce stealthy *AE*s, as the changes required to alter the label will be uniformly destructive. *OFs* allow each pixel (i, j) to have its own kernel $\mathcal{K}^{i,j}$ to customize the filtering at each point. $\mathcal{K}^*$ is the set of all per-pixel $\mathcal{K}^{i,j}$s. The weights $\boldsymbol{\theta}_{\mathcal{K}^*}$ are learned via feedback from a perceptual metric (***lpips***) [56], leading to an assignment that ensures similarity while maximizing the destruction at visually non-critical regions to optimize a specific objective. Note that the ***lpips*** constraint replaces the traditional norm constraint. To guarantee similarity, they also impose geometric constraints via *color kernels* σ_c, similar to ***guided*** filters (details in Appendix I.1).

We subject $\boldsymbol{x}$ to a chain $\prod_{\boldsymbol{OF}}^{B} \equiv \boldsymbol{OF}_{\mathcal{K}^*_1, \boldsymbol{x}, \sigma_{c_1}} \circ \cdots \circ \boldsymbol{OF}_{\mathcal{K}^*_B, \boldsymbol{x}, \sigma_{c_B}}$ of *OFs* similar to ***UnMarker***, replacing the objective pertaining to watermark removal in the filters' weights' learning process with the loss over $\mathcal{M}$. Each *OF* has a kernel set $\mathcal{K}^*_b$ (with wights $\boldsymbol{\theta}_{\mathcal{K}^*_b}$ and shape $M_b \times N_b$), and σ_{c_b}. We optimize:

$$\boldsymbol{x}_{adv} = \underset{\{\boldsymbol{\theta}_{\mathcal{K}^*_b}\}, \boldsymbol{\delta}}{argmin} \begin{bmatrix} \ell_G^{\mathcal{M}}(\prod_{\boldsymbol{OF}}^{B}(\boldsymbol{x} + \boldsymbol{\delta}),\ y) \\ +c \cdot max\{\boldsymbol{lpips}(\boldsymbol{x},\ \prod_{\boldsymbol{OF}}^{B}(\boldsymbol{x} + \boldsymbol{\delta})) - \tau_p, 0\} \end{bmatrix} \tag{7}$$

$\ell_G^{\mathcal{M}}$ denotes any loss as defined in §2. $\boldsymbol{\delta}$ is a modifier that directly optimizes $\boldsymbol{x}$, similar to traditional attacks. *AE*s are generated by manipulating $\boldsymbol{x}$ via $\boldsymbol{\delta}$ and propagating the result through the filters. While direct modifications alone do not cause systemic changes, with *OF*s, they are smoothed over neighbors of the receiving pixels. $\boldsymbol{\delta}$ allows disruptions beyond interpolations. Similar to ***UnMarker***, we chain several *OFs* with different shapes to explore various interpolations. Optimization is iterative (code in Appendix I.3). $max\{\boldsymbol{lpips}(\boldsymbol{x}, \prod_{\boldsymbol{OF}}^{B}(\boldsymbol{x}+\boldsymbol{\delta})) - \tau_p, 0\}$ enforces similarity: If the distance exceeds τ_p, the $\boldsymbol{lpips}$ gradients lower it in the next iteration. Otherwise, it returns 0, minimizing $\ell_G^{\mathcal{M}}$ unconditionally. This gives a solution within the τ_p constraint (violating outputs are discarded), yielding optimal $\{\hat{\boldsymbol{\theta}}_{\mathcal{K}^*_b}\}$, $\hat{\boldsymbol{\delta}}$ s.t. $\boldsymbol{x}_{adv} = \prod_{\boldsymbol{OF}}^{B}(\boldsymbol{x} + \hat{\boldsymbol{\delta}})$, where $\prod_{\boldsymbol{OF}}^{B}$ are the filters with $\{\hat{\boldsymbol{\theta}}_{\mathcal{K}^*_b}\}$.

5.1 *DBP* Against Low-Frequency *AE*s

Our final question is: Can *DBP* be degraded further under *MV*? Based on §5, this is possible with *LF*, which we test in this section. For *LF* (see §5), we use *VGG-LPIPS* [56] as the distance metric with $\tau_p = 0.05$, ensuring imperceptibility [17, 21]. Remaining parameters are similar to ***UnMarker***'s (see Appendix I.2). We use 128 EOT samples for ***CIFAR-10*** (same for label predictions since increasing the sample set size leads to enhanced robustness of *DBP*—see Appendix H). For ***ImageNet***, the numbers are identical to §4.2 (note that the dimensionality of ***ImageNet*** makes larger sample sets prohibitive and our selected set sizes reflect realistic deployments—the largest number of samples that can fit into a modern GPU simultaneously).

Table 3: Performance of *LF* attack under *MV*.

Pur.	Dataset	Models	Cl-Acc %	Rob-Acc %
DiffPure [30]	***ImageNet***	ResNet-50	72.54	**0.00**
		WideResNet-50-2	77.02	**0.00**
		DeiT-S	77.34	**0.00**
	CIFAR-10	WideResNet-28-10	92.19	**2.73**
		WideResNet-70-16	92.19	**3.13**
GDMP [40]	***ImageNet***	ResNet-50	73.05	**0.39**
		WideResNet-50-2	71.88	**0.00**
		DeiT-S	75.00	**0.39**
	CIFAR-10	WideResNet-28-10	93.36	**0.00**
		WideResNet-70-16	92.19	**0.39**

Choice of Perceptual Constraint: We use ***lpips*** with *LF* because it is widely adopted, and prior work [21, 17] provides known thresholds we can reuse for adversarial optimization. For *AE*s, calibrated thresholds are essential; otherwise, an overly permissive constraint can label visibly altered adversarial images as successful, which defeats the point and inflates attack success (i.e., underestimates robustness). Both independently calibrating other non-norm-bounded metrics (e.g., via user studies) and re-running robustness evaluations are time-consuming and require additional resources. Thus, we limit our scope to ***lpips*** but encourage future work to reproduce our experiments with alternative perceptual metrics for which calibrated thresholds may exist or become available.

Results. The *LF* results (using **Full-DiffGrad** for backpropagation) are in Table 3. We also include a ResNet-50 classifier for ***ImageNet***. Not only does it defeat all classifiers completely, leaving the strongest with Rob-Acc of 3.13%, but it also does so in the *MV* setting, where previous attacks fail.

Concluding Remarks. Our findings highlight a limitation in current robustness evaluations, which focus heavily on norm-bounded attacks while overlooking powerful alternatives like low-frequency

(*LF*) perturbations. Though not norm-bounded, *LF* is perceptually constrained—similar to *StAdv* [47], a longstanding benchmark against *DBP* Nie et al. [30]. *LF*'s imperceptibility is guaranteed due to using a perceptual threshold $\tau_p = 0.05$ that has previously been proven to guarantee stealthiness and quality [17, 21]. We include qualitative results in Appendix J confirming this. One may question if existing attacks like *StAdv* can also defeat *DBP* under *MV*, rendering *LF* incremental. We address this in Appendix J, showing *StAdv* fails to produce stealthy *AE*s in this setting. While other techniques may also be adaptable to attacking certain *DBP* variants in the future, *LF* enjoys a solid theoretical foundation such alternatives may lack, limiting their generalizability to all *DBP* defenses.

Future Extensions. While we focus on vision models similar to earlier *DBP* work, both **Theorem 3.1** and **DiffGrad** are modality-agnostic: they depend only on the diffusion dynamics and implementation, not structure. We thus expect the same vulnerabilities for, e.g., speech or video purification. The arguments in support of our *LF* attack's efficacy due to its low-frequency nature also hold across various domains. Yet, the specific implementation that relies on spectral filter networks would require domain-based adaptations to apply it to different tasks (e.g., replacing *2D* Fourier filters with their *1D* counterparts for audio). We therefore urge future work to extend our methods to other modalities.

6 Potential Countermeasures

Adversarial Training (AT). While AT remains a leading defense, it performs poorly on unseen threats. As expected, it fails against our *LF*. On an adversarially trained WideResNet-28-10 for ***CIFAR-10*** (*DiffPure*), *LF* reduces robust accuracy to just 0.78%, confirming this limitation.

SOTA *DBP*: *MimicDiffusion*. One might ask whether recent variants offer improved robustness. Most build incrementally on the foundational defenses in §4, meaning our results broadly generalize. One notable exception is *MimicDiffusion* [35], which generates outputs entirely from noise, using the input only as guidance (see §3.2). Its goal is to preserve semantic content for classification while eliminating adversarial perturbations, and it reports SOTA robustness to adaptive attacks.

However, based on our theoretical analysis, we hypothesize this robustness is illusory—an artifact of improper evaluation and broken gradient flow. Unlike **GDMP**, which involves both direct and guidance paths from $\boldsymbol{x}$ to the loss, *MimicDiffusion* relies solely on guidance. Since guidance gradients require second-order derivatives (disabled by default), the original paper fails to compute meaningful gradients, severely overestimating robustness. We correct this using **DiffGrad**, the first method to properly differentiate through guidance paths. On ***CIFAR-10***, using *AA*-ℓ_∞ (we exclude *LF* for time limitations) with $N = 128$ for both EOT and evaluation, we find that the original Rob-Acc scores of 92.67% (WRN-28-10) and 92.26% (WRN-70-16) collapse under proper evaluation: Wor.Rob drops to 2.73% and 4.3%, respectively. Even under stricter MV.Rob, *MimicDiffusion* performs worse than the standard *DBP* defenses from §4, with accuracies of just 25.78% and 27.73%. This confirms that *MimicDiffusion*'s robustness is not real, and collapses under proper gradients.

Outlook. Our findings highlight the need for more robust defenses. *DBP* remains promising as it improves clean accuracy compared to prior purification methods. Yet, **Theorem 3.1** shows it is vulnerable to adaptive attacks ***when accurate gradients are available***. We believe future work should explore *DBP* variants that relax the assumptions of **Theorem 3.1**. One possible direction is to modify the purification process at inference so that the model follows private stochastic dynamics that are not directly exposed to adversaries. Such approaches, if carefully designed to avoid gradient masking [2], could reduce the transferability of attacker-optimized gradients and increase robustness.

7 Conclusion

We scrutinized *DBP*'s theoretical foundations, overturning its core assumptions. Our analysis of prior findings revealed their reliance on inaccurate gradients, which we corrected to enable reliable evaluations, exposing degraded performance under adaptive attacks. Finally, we evaluated *DBP* in a stricter setup, wherein we found its increased stochasticity leaves it partially immune to norm-bounded *AE*s. Yet, our novel low-frequency approach defeats this defense in both settings. We find current *DBP* is not a viable response to *AE*s, highlighting the need for improvements.

Acknowledgements. This work was supported by NSERC Discovery Grant RGPIN-2020-04722.

References

[1] Brian DO Anderson. Reverse-time diffusion equation models. *Stochastic Processes and their Applications*, 1982.

[2] Anish Athalye, Nicholas Carlini, and David Wagner. Obfuscated gradients give a false sense of security: Circumventing defenses to adversarial examples. In *International Conference on Machine Learning (ICML)*, 2018.

[3] Jacob Buckman, Aurko Roy, Colin Raffel, and Ian J. Goodfellow. Thermometer encoding: One hot way to resist adversarial examples. In *International Conference on Learning Representations (ICLR)*, 2018.

[4] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. In *IEEE Symposium on Security and Privacy (SP)*, 2017.

[5] Nicholas Carlini, Florian Tramer, Krishnamurthy Dj Dvijotham, Leslie Rice, Mingjie Sun, and J Zico Kolter. (Certified!!) adversarial robustness for free! In *International Conference on Learning Representations (ICLR)*, 2023.

[6] Huanran Chen, Yinpeng Dong, Shitong Shao, Zhongkai Hao, Xiao Yang, Hang Su, and Jun Zhu. Your diffusion model is secretly a certifiably robust classifier. In *International Conference on Machine Learning (ICML)*, 2024.

[7] Huanran Chen, Yinpeng Dong, Zhengyi Wang, Xiao Yang, Chengqi Duan, Hang Su, and Jun Zhu. Robust classification via a single diffusion model. In *International Conference on Machine Learning (ICML)*, 2024.

[8] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *arXiv preprint arXiv:1604.06174*, 2016.

[9] Jeremy Cohen, Elan Rosenfeld, and J. Zico Kolter. Certified adversarial robustness via randomized smoothing. In *International Conference on Machine Learning (ICML)*, 2019.

[10] Francesco Croce and Matthias Hein. Reliable evaluation of adversarial robustness with an ensemble of diverse parameter-free attacks. In *International Conference on Machine Learning (ICML)*, 2020.

[11] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2009.

[12] Prafulla Dhariwal and Alexander Quinn Nichol. Diffusion models beat GANs on image synthesis. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[13] Guneet S. Dhillon, Kamyar Azizzadenesheli, Zachary C. Lipton, Jeremy Bernstein, Jean Kossaifi, Aran Khanna, and Animashree Anandkumar. Stochastic activation pruning for robust adversarial defense. In *International Conference on Learning Representations (ICLR)*, 2018.

[14] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations (ICLR)*, 2021.

[15] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations (ICLR)*, 2015.

[16] Chuan Guo, Mayank Rana, Moustapha Cissé, and Laurens van der Maaten. Countering adversarial images using input transformations. In *International Conference on Learning Representations (ICLR)*, 2018.

[17] Qingying Hao, Licheng Luo, Steve TK Jan, and Gang Wang. It's not what it looks like: Manipulating perceptual hashing based applications. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2021.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[19] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[20] Mintong Kang, Dawn Song, and Bo Li. DiffAttack: Evasion attacks against diffusion-based adversarial purification. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2024.

[21] Andre Kassis and Urs Hengartner. UnMarker: A Universal Attack on Defensive Image Watermarking . In *IEEE Symposium on Security and Privacy (SP)*, 2025.

[22] Alex Krizhevsky. Learning multiple layers of features from tiny images, 2009.

[23] Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial examples in the physical world. In *Artificial intelligence safety and security*. Chapman and Hall/CRC, 2018.

[24] Minjong Lee and Dongwoo Kim. Robust evaluation of diffusion-based adversarial purification. In *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2023.

[25] Xuechen Li, Ting-Kam Leonard Wong, Ricky TQ Chen, and David Duvenaud. Scalable gradients for stochastic differential equations. In *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2020.

[26] Yiming Liu, Kezhao Liu, Yao Xiao, ZiYi Dong, Xiaogang Xu, Pengxu Wei, and Liang Lin. Towards understanding the robustness of diffusion-based purification: A stochastic perspective. In *The Thirteenth International Conference on Learning Representations*, 2025. URL `https://openreview.net/forum?id=shqjOIK3SA`.

[27] Keane Lucas, Matthew Jagielski, Florian Tramèr, Lujo Bauer, and Nicholas Carlini. Randomness in ML defenses helps persistent attackers and hinders evaluators. *CoRR*, 2023.

[28] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. In *International Conference on Learning Representations (ICLR)*, 2018.

[29] Dongyu Meng and Hao Chen. MagNet: a two-pronged defense against adversarial examples. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2017.

[30] Weili Nie, Brandon Guo, Yujia Huang, Chaowei Xiao, Arash Vahdat, and Animashree Anandkumar. Diffusion models for adversarial purification. In *International Conference on Machine Learning (ICML)*, 2022.

[31] Yidong Ouyang, Liyan Xie, and Guang Cheng. Improving adversarial robustness through the contrastive-guided diffusion process. In *International Conference on Machine Learning (ICML)*, 2023.

[32] Tianyu Pang, Kun Xu, and Jun Zhu. Mixup inference: Better exploiting mixup to defend adversarial attacks. In *International Conference on Learning Representations (ICLR)*, 2019.

[33] Jonas Rauber, Wieland Brendel, and Matthias Bethge. Foolbox: A python toolbox to benchmark the robustness of machine learning models. In *Reliable Machine Learning in the Wild Workshop, 34th International Conference on Machine Learning*, 2017. URL `http://arxiv.org/abs/1707.04131`.

[34] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *International conference on machine learning (ICML)*, 2015.

[35] Kaiyu Song, Hanjiang Lai, Yan Pan, and Jian Yin. MimicDiffusion: Purifying adversarial perturbation via mimicking clean diffusion model. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2024.

[36] Yang Song, Jascha Sohl-Dickstein, Diederik P. Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *International Conference on Learning Representations (ICLR)*, 2021.

[37] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian J. Goodfellow, and Rob Fergus. Intriguing properties of neural networks. In *International Conference on Learning Representations (ICLR)*, 2014.

[38] Florian Tramèr, Nicholas Carlini, Wieland Brendel, and Aleksander Madry. On adaptive attacks to adversarial example defenses. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[39] Arash Vahdat, Karsten Kreis, and Jan Kautz. Score-based generative modeling in latent space. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2021.

[40] Jinyi Wang, Zhaoyang Lyu, Dahua Lin, Bo Dai, and Hongfei Fu. Guided diffusion model for adversarial purification. *arXiv preprint arXiv:2205.14969*, 2022.

[41] Kaibo Wang, Xiaowen Fu, Yuxuan Han, and Yang Xiang. DiffHammer: Rethinking the robustness of diffusion-based adversarial purification. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2024.

[42] Zekai Wang, Tianyu Pang, Chao Du, Min Lin, Weiwei Liu, and Shuicheng Yan. Better diffusion models further improve adversarial training. In *International Conference on Machine Learning (ICML)*, 2023.

[43] Zhou Wang, Alan C Bovik, Hamid R Sheikh, and Eero P Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 2004.

[44] Eric Wong, Leslie Rice, and J. Zico Kolter. Fast is better than free: Revisiting adversarial training. In *International Conference on Learning Representations (ICLR)*, 2020.

[45] Quanlin Wu, Hang Ye, and Yuntian Gu. Guided diffusion model for adversarial purification from random noise. *arXiv preprint arXiv:2206.10875*, 2022.

[46] Chang Xiao, Peilin Zhong, and Changxi Zheng. Enhancing adversarial defense by k-winners-take-all. In *International Conference on Learning Representations (ICLR)*, 2020.

[47] Chaowei Xiao, Jun-Yan Zhu, Bo Li, Warren He, Mingyan Liu, and Dawn Song. Spatially transformed adversarial examples. In *International Conference on Learning Representations (ICLR)*, 2018.

[48] Chaowei Xiao, Zhongzhu Chen, Kun Jin, Jiongxiao Wang, Weili Nie, Mingyan Liu, Anima Anandkumar, Bo Li, and Dawn Song. DensePure: Understanding diffusion models towards adversarial robustness. In *Workshop on Trustworthy and Socially Responsible Machine Learning, NeurIPS*, 2022.

[49] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan Yuille. Mitigating adversarial effects through randomization. In *International Conference on Learning Representations (ICLR)*, 2018.

[50] Yuzhe Yang, Guo Zhang, Dina Katabi, and Zhi Xu. ME-Net: Towards effective adversarial robustness with matrix estimation. In *International Conference on Machine Learning (ICML)*, 2019.

[51] Jongmin Yoon, Sung Ju Hwang, and Juho Lee. Adversarial purification with score-based generative models. In *International Conference on Machine Learning (ICML)*, 2021.

[52] Sergey Zagoruyko. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

[53] Boya Zhang, Weijian Luo, and Zhihua Zhang. Purify++: Improving diffusion-purification with advanced diffusion models and control of randomness. *arXiv preprint arXiv:2310.18762*, 2023.

[54] Hongyang Zhang, Yaodong Yu, Jiantao Jiao, Eric P. Xing, Laurent El Ghaoui, and Michael I. Jordan. Theoretically principled trade-off between robustness and accuracy. In *International Conference on Machine Learning (ICML)*, 2019.

[55] Jiawei Zhang, Zhongzhu Chen, Huan Zhang, Chaowei Xiao, and Bo Li. DiffSmooth: Certifiably robust learning via diffusion models and local smoothing. In *USENIX Security Symposium (USENIX Security)*, 2023.

[56] Richard Zhang, Phillip Isola, Alexei A Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification:

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We provide a "Limitations" section in Appendix A.

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes]

Justification: The assumptions underlying Theorem 3.1 are in Section 3.1 and the proof in Appendix C.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: Reproducibility information is given in Section 4 and Appendices I.2 and E.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: Our code is available here: `https://github.com/andrekassis/DiffBreak`. We use public datasets CIFAR-10 and ImageNet.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://nips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: Experiment settings and details are given in Section 4 and Appendices I.2 and E.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No]

   Justification: Computing error bars would have been too computationally expensive given our limited computational resources and the extensive scope of our experiments, far exceeding typical settings in relevant prior work.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Information about compute resources is given in Section 4.2. Experiments are conducted on a 40GB NVIDIA A100 GPU.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification:

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Broader impacts are discussed in Appendix A.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification:

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We use public datasets CIFAR-10 and ImageNet and give corresponding citations.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: Our code repository (`https://github.com/andrekassis/DiffBreak`) is well documented.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification:

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification:

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

    Answer: [NA]

    Justification:

    Guidelines:

    - The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

A Broader Impact & Limitations

This work advances adversarial robustness by exposing a fundamental vulnerability in Diffusion-Based Purification (*DBP*). We show that adaptive attacks do not merely circumvent *DBP* but repurpose it as an adversarial generator, invalidating its theoretical guarantees. While this insight highlights a critical security risk, it also provides a foundation for designing more resilient purification strategies. To facilitate rigorous and reproducible evaluation, we introduce *DiffBreak*, an open-source toolkit providing the first reliable gradient module for *DBP*-based defenses, in addition to a wide array of attack implementations, including our low-frequency (*LF*) strategy that outperforms existing approaches, seamlessly applicable to a broad range of classifiers. Similar to existing adversarial attack toolkits (e.g., *Foolbox* [33], *AutoAttack* [10]), our framework is intended for research and defensive purposes. While this tool enhances robustness assessments, we acknowledge its potential misuse in adversarial applications. To mitigate risks, we advocate its responsible use for research and defensive purposes only. Finally, we urge developers of *ML* security systems to integrate our findings to design more resilient defenses against adaptive attacks. We encourage future research to explore fundamentally secure purification methods that are inherently resistant to manipulations.

Limitations. Our work focuses on dissecting the robustness of *DBP* defenses, both theoretically and empirically, with an emphasis on attack strategy. While we briefly consider existing mitigation techniques, future work should explore how diffusion models can be more effectively leveraged for defense. Classifiers could also be adversarially trained on *LF* perturbations to improve robustness. Yet, the vast hyperparameter and architecture space of *LF* potentially creates an effectively infinite threat surface, limiting the practicality of adversarial training. We leave this to future investigation.

While we do not report formal error bars or confidence intervals, this decision aligns with standard practice in the literature (e.g., [20, 26]) due to the high computational cost of gradient-based *DBP* attacks. Instead, we perform extensive experiments and also provide repeated empirical trials and variance analyses in Appendix D.2 that confirm our conclusions are robust, despite the absence of formal statistical intervals.

Although our main text focuses on conceptual and empirical issues in *DBP*, Appendix D.1.3 and Appendix D.3 provide theoretical and asymptotic cost analysis, and Appendix H presents latency measurements and real-world speedup comparisons. Our module is efficient, scales linearly with diffusion steps, and outperforms prior implementations in both latency and throughput. Thus, while cost is not emphasized in the main paper, it is extensively addressed and poses no limitation in practice. Nonetheless, even with our efficient implementation, the dominant cost from *DBP* itself is inherited by attacks and robustness evaluations, rendering them costly regardless. That said, *DBP*'s practicality may improve with advanced hardware or more efficient schemes in the future.

B Equivalence of DDPM and VP-SDE

An alternative to the *continuous-time* view described in §2 (i.e., **VP-SDE**) is *Denoising diffusion probabilistic modeling* (**DDPM**) [19, 34], which considers a *discrete-time* framework (*DM*) where the forward and reverse passes are characterized by a maximum number of steps T. Here, the forward pass is a Markov chain:

$$\boldsymbol{x}_i = \sqrt{1-\beta_i}\boldsymbol{x}_{i-1} + \sqrt{\beta_i}\boldsymbol{z}_i$$

where $\boldsymbol{z}_i \sim \mathcal{N}(\mathbf{0},\ \boldsymbol{I}_d)$ and β_i is a small positive noise scheduling constant. Defining $dt = \frac{1}{T}$, we know due to Song et al. [36] that when $T \longrightarrow \infty$ (i.e, $dt \longrightarrow 0$, which is the effective case of interest), this converges to the *SDE* in *eq. (1)* (with the drift and diffusion function $\boldsymbol{f}$ and g described in §2). The reverse pass is also a Markov chain given as:

$$d\hat{\boldsymbol{x}} = \hat{\boldsymbol{x}}_{i-1} - \hat{\boldsymbol{x}}_i = \frac{1}{\sqrt{1-\beta_i}}((1-\sqrt{1-\beta_i})\hat{x}_i + \beta_i \boldsymbol{s}_\theta(\hat{\boldsymbol{x}}_i, i)) + \sqrt{\beta_i}\boldsymbol{z}_i \tag{8}$$

When $T \longrightarrow \infty$, $d\hat{\boldsymbol{x}}$ converges to *eq. (4)* (see [36]). Thus, the two views are effectively equivalent. Accordingly, we focus on the continuous view, which encompasses both frameworks. For discrete time, $\boldsymbol{x}(t)$ will be used to denote $\boldsymbol{x}_{|\frac{t}{dt}|}$.

C Proof of Theorem 3.1

Theorem 3.1. The adaptive attack **optimizes the entire reverse diffusion process**, modifying the parameters $\{\theta^t_{\boldsymbol{x}}\}_{t\leqslant t*}$ such that the output distribution $\hat{\boldsymbol{x}}(0) \sim \mathit{DBP}^{\{\theta^t_{\boldsymbol{x}}\}}(\boldsymbol{x})$, where $\mathit{DBP}^{\{\theta^t_{\boldsymbol{x}}\}}(\boldsymbol{x})$ is the *DBP* pipeline with the score model's weights $\{\theta^t_{\boldsymbol{x}}\}_{t\leqslant t*}$ adversarially aligned. That is, the adversary **implicitly controls the purification path** and optimizes the weights $\{\theta^t_{\boldsymbol{x}}\}_{t\leqslant t*}$ to maximize:

$$\max_{\{\theta^t_{\boldsymbol{x}}\}_{t\leqslant t*}} \mathbb{E}_{\hat{\boldsymbol{x}}(0)\sim \mathit{DBP}^{\{\theta^t_{\boldsymbol{x}}\}}(\boldsymbol{x})}[\Pr(\neg y|\hat{\boldsymbol{x}}(0))].$$

Proof. The adversary aims to maximize:

$$\mathcal{Q}(\boldsymbol{x}) \equiv \mathbb{E}_{\hat{\boldsymbol{x}}(0)\sim \mathit{DBP}^{\{\theta^t\}}(\boldsymbol{x})}[\Pr(\neg y|\hat{\boldsymbol{x}}(0))]$$

Expanding this expectation yields the following alternative representations:

$$\begin{aligned}
\mathcal{Q}(\boldsymbol{x}) &= \int \Pr(\neg y|\hat{\boldsymbol{x}}(0))p(\hat{\boldsymbol{x}}(0))d\hat{\boldsymbol{x}}(0) \\
&= \int \Pr(\neg y|\hat{\boldsymbol{x}}(0)) \int p(\hat{\boldsymbol{x}}(0)|\hat{\boldsymbol{x}}_{t*:-dt})p(\hat{\boldsymbol{x}}_{t*:-dt})d\hat{\boldsymbol{x}}_{t*:-dt}d\hat{\boldsymbol{x}}(0) \\
&= \int \Pr(\neg y|\hat{\boldsymbol{x}}(0))p(\hat{\boldsymbol{x}}_{t*:0})d\hat{\boldsymbol{x}}_{t*:0}.
\end{aligned}$$

The first transition is due to the definition of expectation, the second follows from the definition of marginal probability, and the final transition replaces the joint density function with its compact form.

Since this formulation abstracts away $\boldsymbol{x}$ and $\{\theta^t_{\boldsymbol{x}}\}_{t\leqslant t*}$, we explicitly include them as $p(\hat{\boldsymbol{x}}_{t*:0}) \equiv p(\hat{\boldsymbol{x}}_{t*:0}|\boldsymbol{x}, \{\theta^t_{\boldsymbol{x}}\}_{t\leqslant t*})$. For notation simplicity, we define $p_{\theta_{\boldsymbol{x}}}(\hat{\boldsymbol{x}}_{t*:0}|\boldsymbol{x}) \equiv p(\hat{\boldsymbol{x}}_{t*:0}|\boldsymbol{x},\ \{\theta^t_{\boldsymbol{x}}\}_{t\leqslant t*})$. Thus, the final optimization objective is:

$$\mathcal{Q}(\boldsymbol{x}) = \int \Pr(\neg y|\hat{\boldsymbol{x}}(0))p_{\theta_{\boldsymbol{x}}}(\hat{\boldsymbol{x}}_{t*:0}|\boldsymbol{x})d\hat{\boldsymbol{x}}_{t*:0}.$$

Since we assume smoothness ($\mathcal{C}^2$), we interchange gradient and integral:

$$\nabla_{\boldsymbol{x}}[\mathcal{Q}(\boldsymbol{x})] = \int \Pr(\neg y|\hat{\boldsymbol{x}}(0))\nabla_{\boldsymbol{x}} p_{\theta_{\boldsymbol{x}}}(\hat{\boldsymbol{x}}_{t*:0}|\boldsymbol{x})d\hat{\boldsymbol{x}}_{t*:0}.$$

where the last step is because $\Pr(\neg y|\hat{\boldsymbol{x}}(0))$ does not depend on $\boldsymbol{x}$, but only on $\hat{\boldsymbol{x}}(0)$ which is independent of $\boldsymbol{x}$ (though its probability of being the final output of *DBP* is a function of $\boldsymbol{x}$). Optimizing the objective via gradient ascent relies on altering *DBP*'s output distribution alone, evident in its gradient where $\Pr(\neg y|\hat{\boldsymbol{x}}(0))$ assigned by the classifier is a constant with its gradient ignored. While we lack direct access to the gradients of the probabilistic paths above, we may still attempt to solve the optimization problem.

Recall that by definition, $\forall \boldsymbol{u}\ \mathcal{M}^y(\boldsymbol{u}) = \Pr(y|\boldsymbol{u})$. Thus, the required gradient can also be expressed as:

$$\nabla_{\boldsymbol{x}}\big[\mathcal{Q}(\boldsymbol{x})\big] = \nabla_{\boldsymbol{x}}\Bigg[\int \Pr(\neg y|\hat{\boldsymbol{x}}(0))p_{\theta_{\boldsymbol{x}}}(\hat{\boldsymbol{x}}_{t*:0}|\boldsymbol{x})d\hat{\boldsymbol{x}}_{t*:0}\Bigg] =$$

$$\nabla_{\boldsymbol{x}}\Bigg[\mathop{\mathbb{E}}_{p_{\theta_{\boldsymbol{x}}}(\hat{\boldsymbol{x}}_{t*:0}|\boldsymbol{x})}[1-\mathcal{M}^y(\hat{\boldsymbol{x}}(0))]\Bigg] = -\nabla_{\boldsymbol{x}}\Bigg[\mathop{\mathbb{E}}_{p_{\theta_{\boldsymbol{x}}}(\hat{\boldsymbol{x}}_{t*:0}|\boldsymbol{x})}[\mathcal{M}^y(\hat{\boldsymbol{x}}(0))]\Bigg]$$

Yet, this expectation is over probabilistic paths whose randomness is due to the noise $\boldsymbol{\epsilon}$ of the forward pass and the Brownian motion $d\bar{\boldsymbol{w}}_t$ at each reverse step t that are independent. Hence, by the law of the unconscious statistician:

$$-\nabla_{\boldsymbol{x}}\Bigg[\mathop{\mathbb{E}}_{p_{\theta_{\boldsymbol{x}}}(\hat{\boldsymbol{x}}_{t*:0}|\boldsymbol{x})}[\mathcal{M}^y(\hat{\boldsymbol{x}}(0))]\Bigg] = -\nabla_{\boldsymbol{x}}\Bigg[\mathop{\mathbb{E}}_{p_r(\boldsymbol{\epsilon},d\bar{\boldsymbol{w}}_{t*},\ldots,d\bar{\boldsymbol{w}}_0)}[\mathcal{M}^y(\hat{\boldsymbol{x}}(0))]\Bigg]$$

where $p_r(\boldsymbol{\epsilon}, d\bar{\boldsymbol{w}}_{t*}, ..d\bar{\boldsymbol{w}}_0)$ denotes the joint distribution of the noise $\boldsymbol{\epsilon}$ and the Brownian motion vectors in the reverse pass. Note that $\hat{\boldsymbol{x}}(0)$ on the RHS denotes the output obtained by invoking the

DBP pipeline with some assignment of these random vectors on the sample $\boldsymbol{x}$. As earlier, we can interchange the derivation and integration, obtaining:

$$- \mathop{\mathbb{E}}_{p_r(\boldsymbol{\epsilon}, d\bar{\boldsymbol{w}}_{t*}, \ldots, d\bar{\boldsymbol{w}}_0)} \left[\nabla_{\boldsymbol{x}} [\mathcal{M}^y(\hat{\boldsymbol{x}}(0))] \right].$$

Let $\boldsymbol{G}^{\boldsymbol{x}}$ denote the random variable that is assigned values from $\nabla_{\boldsymbol{x}} \mathcal{M}^y(\hat{\boldsymbol{x}}(0))$, where $\hat{\boldsymbol{x}}(0)$ is as described above, and denote its covariance matrix by $\Sigma_{\boldsymbol{G}^{\boldsymbol{x}}}$. Essentially, we are interested in $\mathbb{E}[\boldsymbol{G}^{\boldsymbol{x}}]$. If we define $\widetilde{\nabla}_{\boldsymbol{x}}$ as:

$$\widetilde{\nabla}_{\boldsymbol{x}} = \frac{1}{N} \sum_{n=1}^{N} \nabla_{\boldsymbol{x}} [\mathcal{M}^y(\hat{\boldsymbol{x}}(0)_n)]$$

where each $\hat{\boldsymbol{x}}(0)_n$ is the output of *DBP* invoked with a certain random path $(\boldsymbol{\epsilon}^n, d\bar{\boldsymbol{w}}^n_{t*}, ..d\bar{\boldsymbol{w}}^n_0) \overset{i.i.d}{\sim} p(\boldsymbol{\epsilon}, d\bar{\boldsymbol{w}}_{t*}, ..d\bar{\boldsymbol{w}}_0)$ and N is a sufficiently large number of samples. Then due to the central limit theorem, $\widetilde{\nabla}_{\boldsymbol{x}} \longrightarrow \mathcal{N}(\mathbb{E}[\boldsymbol{G}^{\boldsymbol{x}}], \frac{\Sigma_{\boldsymbol{G}^{\boldsymbol{x}}}}{N})$. That is, propagating multiple copies through *DBP* and then averaging the loss gradients computes the required gradient for forcing *DBP* to alter its output distribution. The larger the number N of samples is, the lower the variance of the error, as can be seen above. Note that the adaptive attack operates exactly in this manner, assuming it uses $\mathcal{M}^y$ as the loss it minimizes (but the soundness of the approach generalizes intuitively to any other loss with the same objective over the classifier's logits), proving our claim. □

D Details on Our DiffGrad Gradient Module

In Appendix D.1, we identify key implementation challenges in prior work, detailing how **DiffGrad** systematically resolves each one. Then, in Appendix D.2, we empirically demonstrate their impact on *DBP*'s gradients when left unaddressed, reinforcing their practical significance and validating the need for our reliable module. Appendix D.3 presents the pseudo-code for our memory-efficient checkpointing algorithms, offering a transparent blueprint for reproducible and correct gradient-enabled purification. Finally, in Appendix D.4, we verify the correctness of our implementation, confirming that the gradients computed by **DiffGrad** match those produced by a true differentiable purification pipeline.

D.1 The Challenges of Applying Checkpointing to *DBP* and How DiffGrad Tackles Them

D.1.1 High-Variance Gradients

The challenge of high-variance gradients due to insufficient EOT samples is introduced in §3.2. We defer its empirical analysis to Appendix D.2, where we quantify the gradient variability under different sample counts. Practical implications for performance and throughput—enabled by **DiffGrad**'s design—are explored in the ablation study in Appendix H. Accordingly, we focus below on the remaining implementation issues.

D.1.2 Time Consistency

Rounding issues may emerge when implementing checkpointing over *torchsde* solvers: Given the starting time t*, the solver internally converts it into a *PyTorch* tensor, iteratively calculating the intermediate outputs by adding negative time increments to this tensor. On the other hand, checkpointing requires re-calculating the intermediate steps' samples during back-propagation. If this code is oblivious to *torchsde*'s conversion of the initial time into a *PyTorch* tensor, it will continue to treat it as a floating point number, updating it with increments of the time step to obtain each intermediate t used to re-compute the dependencies. The *PyTorch* implementation does not aim for 100% accuracy, leading to minute discrepancies in the current value of t compared to pure floating-point operations. These inaccuracies accumulate over the time horizon, potentially severely affecting the gradients. Instead, we ensure both the solver and checkpointing code use the same objects (either floating points or tensors).

D.1.3 Reproducing Stochasticity

CuDNN-Induced Nondeterminism. We found that even in deterministic purification pipelines, subtle nondeterminism from PyTorch's backend can cause discrepancies between forward and backward propagations. Specifically, CuDNN may select different convolution kernels during the recomputation of intermediate states in checkpointing, leading to minute numerical drift.

To eliminate this, *DiffBreak* explicitly forces CuDNN to behave deterministically, guaranteeing consistent kernel selection and numerical fidelity. While this source of error is subtle, it is measurable: for *DiffPure* with $t^* = 0.1$ and $dt = 0.001$, we observed raw and relative gradient discrepancies of $1e-4$ and $7.28e-5$ when all other issues have been addressed. With this fix, the discrepancy dropped to exactly zero. Although less severe than other sources of error (e.g., time inconsistency or missing gradient paths), this level of precision is critical in setups where gradients are small to prevent sign changes that can affect optimization trajectories. Our inspection of prior work's code reveals this drift was not previously addressed.

Noise Sampler for Guidance Robustness. While the contribution of the stochastic component of $d\hat{\boldsymbol{x}}$ often cancels analytically in vanilla **VP-SDE**-style purification (see §2), meaning it does not affect the gradient and therefore is not required to be accurately reproduced during checkpointing, this is not guaranteed in all variants. In particular, many guidance-based schemes—see §3.2—integrate stochastic terms within the guidance function itself. That is, the guidance function may incorporate stochasticity. When this occurs, gradients become sensitive to noise realizations, and failure to preserve them could break correctness. Yet, standard checkpointing only stores the intermediate $\hat{\boldsymbol{x}}(t)$s. Hence, $\hat{\boldsymbol{x}}(t+dt)$ will differ between the propagation phases as $d\hat{\boldsymbol{x}}$ is computed via different random variables.

We restructure the logic computing $d\hat{\boldsymbol{x}}$: We define a function $\boldsymbol{calc\_dx}$ that accepts the noise as an external parameter and utilize a *Noise Sampler* $\boldsymbol{NS}$, initialized upon every forward propagation. For each t, $\boldsymbol{NS}$ returns all the necessary random noise vectors to pass to $\boldsymbol{calc\_dx}$. Instead of $\hat{\boldsymbol{x}}(0)$ only, forward propagation also outputs $\boldsymbol{NS}$ and a state $\boldsymbol{S}$ containing all $\hat{\boldsymbol{x}}(t)$s. These objects are used to restore the path in backpropagation. This design also future-proofs *DiffBreak* for emerging guided or hybrid purifiers: any randomness on which the guidance depends is captured by $\boldsymbol{NS}$ and replayed exactly during differentiation.

The memory cost of our design is $\mathcal{O}(\frac{t^*}{|dt|})$, storing only $\boldsymbol{NS}$, and all $\hat{\boldsymbol{x}}(t)$s, each with a negligible footprint ($\mathcal{O}(1)$) compared to graph dependencies [20].

D.1.4 Guidance Gradients

In schemes that involve guidance, it is typically obtained by applying a function $\boldsymbol{g_{fn}}$ to $\hat{\boldsymbol{x}}(t)$ at each step (note that $\boldsymbol{g_{fn}}$ may involve stochasticity—see Appendix D.1.3—as is often the case in *GDMP* [40]). For instance, in *Guided*-**DDPM** [40], the original sample $\boldsymbol{x}$ is used to influence the reverse procedure to retain key semantic information, allowing for an increased budget t^* to better counteract adversarial perturbations. Effectively, it modifies *eq. (8)* describing the reverse pass of **DDPM** as:

$$d\hat{\boldsymbol{x}} = \frac{1}{\sqrt{1-\beta_i}}((1-\sqrt{1-\beta_i})\hat{\boldsymbol{x}}_i + \beta_i \boldsymbol{s}_\theta(\hat{\boldsymbol{x}}_i, i)) - s\beta_i \nabla_{\hat{\boldsymbol{x}}_i} \boldsymbol{GC}(\hat{\boldsymbol{x}}_i,\ \boldsymbol{x}) + \sqrt{\beta_i}\boldsymbol{z}_i$$

where $\boldsymbol{GC}$ is a guidance condition (typically a distance metric), each step minimizes by moving in the opposite direction of its gradient, while the scale s controls the guidance's influence.

That is, in the specific case of *Guided*-**DDPM**, $\boldsymbol{g_{fn}}(\hat{\boldsymbol{x}}(t)) \equiv -\nabla_{\hat{\boldsymbol{x}}(t)} \boldsymbol{GC}(\hat{\boldsymbol{x}}(t),\ \boldsymbol{x})$, where $\boldsymbol{GC}$ is a distance metric. Nonetheless, other choices for $\boldsymbol{g_{fn}}$ may be employed in general. Typically, as the goal of the guidance is to ensure key information from the original sample $\boldsymbol{x}$ is retained, $\boldsymbol{g\_fn}$ will also *directly* involve this $\boldsymbol{x}$ in addition to $\hat{\boldsymbol{x}}(t)$ (e.g., $\boldsymbol{GC}$ above). Yet, a naive implementation would back-propagate the gradients to $\boldsymbol{x}$ by only considering the path through $\hat{\boldsymbol{x}}(t)$. Yet, when $\boldsymbol{g\_fn}$ relies on a $\boldsymbol{guide}$ constructed from $\boldsymbol{x}$ to influence $\hat{\boldsymbol{x}}(t)$ (e.g., $\boldsymbol{guide} \equiv \boldsymbol{x}$ in *Guided*-**DDPM**, it creates additional paths from $\boldsymbol{x}$ to the loss through this $\boldsymbol{guide}$ at each step t. Accordingly, **DiffGrad** augments the process to include the gradients due to these paths. In the general case, this $\boldsymbol{guide}$ may not be identical to $\boldsymbol{x}$ but can rather be computed based on $\boldsymbol{x}$ or even completely independent (in which case no guidance gradients are collected). **DiffGrad** captures this nuance through an abstract function $\boldsymbol{g\_aux}$ that, given $\boldsymbol{x}$, outputs the $\boldsymbol{guide}$. Similar to *eq. (6)*, we have that for each t, the

gradient of any function F applied to $\hat{\boldsymbol{x}}(0)$ w.r.t. $\boldsymbol{guide}$ due to the path from $\hat{\boldsymbol{x}}(t+dt)$ to $\hat{\boldsymbol{x}}(0)$ is given by:

$$\nabla^t_{\boldsymbol{guide}} F = \nabla_{\boldsymbol{guide}} \langle \hat{\boldsymbol{x}}(t+dt), \nabla_{\hat{\boldsymbol{x}}(t+dt)} F \rangle \tag{9}$$

as $\hat{\boldsymbol{x}}(t+dt)$ is a function of $\boldsymbol{guide}$. Recall that we are interested in F, which is the loss function over the classifier's output on $\hat{\boldsymbol{x}}(0)$. The gradient of F w.r.t. $\boldsymbol{guide}$ is a superposition of all these paths' gradients, given as:

$$\nabla_{\boldsymbol{guide}} F = \sum_t \nabla^t_{\boldsymbol{guide}} F \tag{10}$$

By the chain rule, F's gradient w.r.t $\boldsymbol{x}$ due to the guidance paths, which we denote as $\nabla^g_{\boldsymbol{x}} F$, is:

$$\nabla^g_{\boldsymbol{x}} F = \nabla_{\boldsymbol{x}} \langle \boldsymbol{guide}, \nabla_{\boldsymbol{guide}} F \rangle \tag{11}$$

As a convention, for an unguided process or when $\boldsymbol{guide}$ and $\boldsymbol{x}$ are not related, we define the gradient returned from *eq. (11)* as $\nabla^g_{\boldsymbol{x}} F \equiv \mathbf{0}$, preserving correctness in general. Since $\boldsymbol{x}$ in this guided scenario traverses both the guidance and standard purification paths, the final gradient is the sum of both components.

Finally, automatic differentiation engines, by default, generate gradients without retaining dependencies on the inputs that produced them. Thus, when the guidance itself is in the form of a gradient as in the case of *Guided*-**DDPM** or other potential alternatives, its effects will not be back-propagated to $\boldsymbol{x}$ through any of the two paths described above, despite our proposed extensions. **DiffGrad** alters this behavior, retaining the dependencies of such gradient-based guidance metrics as well.

D.1.5 Proper Use of The Surrogate Method [24]

As noted in §3.2, **DiffHammer** [41] adopts the **surrogate** method (originally proposed by Lee and Kim [24]) for gradient computation, as acknowledged in their Appendix C.1.1. This approach approximates *DBP*'s gradients by replacing the standard fine-grained reverse process (e.g., $dt = 0.001$) with a coarser one (e.g., $\bar{dt} = 0.01$), thereby enabling memory-efficient differentiation using standard autodiff. For instance, when using a diffusion time $t^* = 0.1$ for *DBP* (see §2), the fine-grained process would require 100 steps, which is computationally infeasible (memory-exhaustive) for gradient computation without checkpointing. In contrast, the coarse-grained process would only require 10 steps, which is possible.

Specifically, the forward pass (diffusion) is computed using the closed-form solution from eq. (2), while the reverse pass uses $\bar{dt}$, enabling efficient gradient computation. Notably, this replacement only occurs for the purpose of updating the attack sample (i.e., computing the gradients) and happens during both forward and backpropagation so that the two phases match. In contrast, when evaluating the generated *AE*, it is purified via the standard fine-grained process that uses dt, as this is the true (full) procedure the model owner (defender) would execute. Although the surrogate process yields approximate gradients for a slightly different process, Lee and Kim [24] found it effective and more accurate than other approximations like the **adjoint** method [25] (despite remaining slightly suboptimal—see Appendix F).

However, a manual inspection of **DiffHammer**'s code[3] reveals that their implementation of this surrogate method contains a critical issue. Rather than computing the reverse process with the coarse $\bar{dt}$ as intended for attack optimization during both forward and backpropagation, they first run the reverse pass during forward propagation with the standard dt, storing intermediate states. Then, during backpropagation, they attempt to backpropagate gradients using checkpoints spaced at $\bar{dt}$ intervals, but instead reuse the stored dt-based forward states. This leads to a mismatch: gradients intended for steps of size $\bar{dt}$ are applied to states computed with dt, effectively disconnecting the computation graph and introducing significant gradient errors. In practice, this behaves like a hybrid between checkpointing and **BPDA** [2], where gradients are approximated around fixed anchor points without correctly tracking the reverse dynamics.

As we show in §4.2, this implementation leads to significant attack performance degradation compared to the full gradients. It illustrates the fragility of gradient computation in diffusion attacks and further motivates the need for our reliable **DiffGrad** module.

[3] `https://github.com/Ka1b0/DiffHammer` (accessed in May 2025)

D.2 Demonstrating The Effects of Incorrect Backpropagation

We empirically demonstrate the impact of the identified issues on *DBP*'s gradients in Fig. 2 and briefly explain each result below. As numerical results for CuDNN-induced nondeterminism (i.e., unhandled stochasticity) are already presented in Appendix D.1.3, we focus here on the remaining four issues detailed in Appendix D.1:

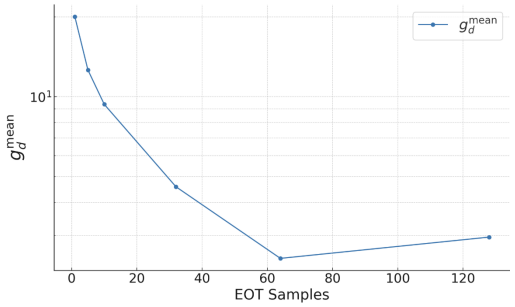
(a) Effect of insufficient EOT samples. Fig.2a illustrates the impact of using too few EOT samples when computing gradients—see§3.2 and Appendix D.1.1. While prior works [41, 20, 24, 26, 30] typically use 10–20 samples, such low counts, while somewhat effective, may introduce significant variance, resulting in noisy gradients and suboptimal attacks.

To quantify this, we run the following experiment on a ***CIFAR-10*** sample $\boldsymbol{x}$ purified with *DiffPure* (t^*=0.1, dt=10^{-3}). For each EOT count N (shown on the x-axis), we generate N purified copies, compute the average loss, and backpropagate to obtain the EOT gradient. This is repeated 20 times to yield 20 gradients $g_1^N, \ldots, g_{20}^N$ for each N. We then define:

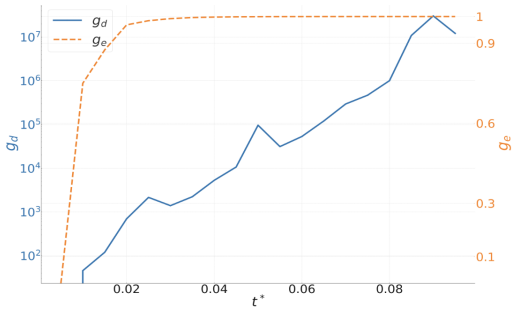
$$g_d^{mean}(N) = \frac{1}{20} \sum_{i=1}^{20} \underset{j}{max} \, \| \boldsymbol{g}_i^N - \boldsymbol{g}_j^N \|_2$$

This metric reflects the worst-case deviation between gradients under repeated EOT sampling. By the central limit theorem (see Appendix C), the variance of EOT gradients decays with N, and thus $g_d^{\text{mean}}(N) \to 0$ as $N \to \infty$. The curve in Fig. 2a captures this decay and lets us identify a threshold where variance becomes acceptably low.

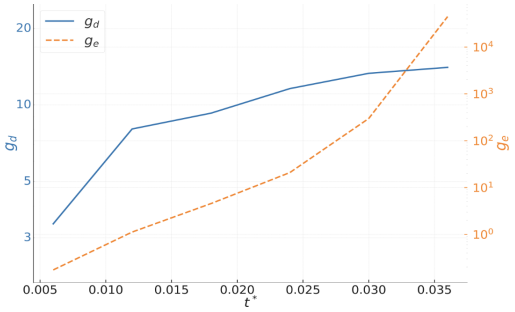
The plot confirms that gradient variance decreases sharply with the number of EOT samples. While $N = 10$—the typical choice in prior work—is somewhat effective and viable under resource constraints, it remains suboptimal. We observe that variance continues to drop until around $N = 64$, after which it plateaus. This suggests that although larger N values like 128 yield marginally better stability, the bulk of the benefit is realized by $N = 32$–64. Thus, $N = 10$ is a practical baseline, but $N \geqslant 64$ should be preferred when accuracy is critical and compute allows, coroborating our claims in §3.2.



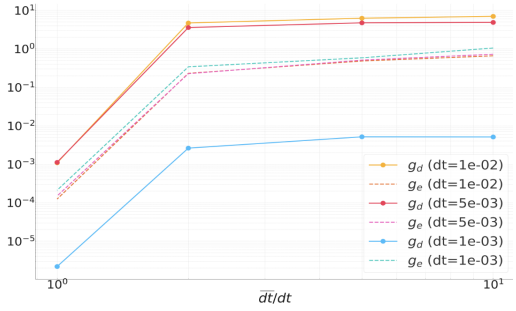
(a) Effect of insufficient EOT samples



(b) Effect of time inconsistency



(c) Effect of ignoring guidance gradients



(d) Effect of incorrect surrogate gradients

Figure 2: Effects of the identified issues in *DBP*'s backpropagation. Each subfigure visualizes a specific error source.

(b) Effect of time inconsistency. As detailed in §3.2 and Appendix D.1.2, rounding inconsistencies can emerge when using checkpointing with *torchsde* solvers. These stem from how the solver internally handles time as a *PyTorch* tensor, which may diverge subtly from floating-point representations if not carefully synchronized. While such discrepancies are negligible for short purification paths (small t^*), they accumulate significantly over longer trajectories—particularly at standard *DBP* settings like t^*=0.1 with dt=10^{-3} (i.e., 100 reverse steps), as in *DiffPure* on ***CIFAR-10***.

To quantify this effect, we purify a fixed ***CIFAR-10*** sample $\boldsymbol{x}$ using values of t^* from 0.005 to 0.1 in steps of 0.005. For each t^*, we compute gradients using **DiffGrad** under two conditions: (1) without correcting the inconsistency (yielding $\boldsymbol{g}_{nf}$), and (2) with the issue fixed (yielding $\boldsymbol{g}_f$), both using the same stochastic path. We then compute two metrics: (1) $g_d = \|\boldsymbol{g}_f - \boldsymbol{g}_{nf}\|_2$ (absolute error), and (2) $g_e = \frac{\|\boldsymbol{g}_f - \boldsymbol{g}_{nf}\|_2}{\|\boldsymbol{g}_f\|_2}$ (relative error). These quantify the divergence introduced by uncorrected rounding—both in magnitude and in proportion to the true gradient—and illustrate its increasing severity with longer reverse trajectories.

Fig. 2b confirms that even minute rounding discrepancies in time handling, if uncorrected, can completely corrupt gradients over longer diffusion trajectories. Both the absolute error g_d (blue) and the relative error g_e (orange) grow rapidly with t^*. By $t^* \geqslant 0.03$, g_e exceeds 90%, and for the typical setting of $t^* = 0.1$, the relative error reaches 100%, indicating that the gradients are nearly orthogonal to the correct direction. This validates our claim that unpatched checkpointing introduces severe errors and demonstrates that our **DiffGrad** fix is essential for correct gradient computation in practical *DBP* setups.

(c) Effect of ignored guidance gradients. We evaluate the impact of neglecting guidance gradients, a critical problem in current attacks against *Guided DBP* defenses (e.g., *GDMP*, *MimicDiffusion*)—see §3.2 and Appendix D.1.4. As explained earlier, guidance introduces additional gradient paths from the input $\boldsymbol{x}$ to $\hat{\boldsymbol{x}}(t)$, typically requiring second-order derivatives to capture. Existing approaches ignore these paths, leading to incomplete gradients and suboptimal attacks. To our knowledge, **DiffGrad** is the first system to correctly handle them.

To quantify this effect, we purify a fixed ***CIFAR-10*** sample $\boldsymbol{x}$ using *GDMP* (setup in Appendix E) while varying the purification horizon $t^* \in \{0.006, 0.012, \ldots, 0.036\}$. For each t^*, we compute gradients using **DiffGrad** with and without accounting for guidance—yielding $\boldsymbol{g}_f$ and $\boldsymbol{g}_{nf}$, respectively—under the same stochastic path. We then compute the absolute and relative gradient error (g_d, g_e) as in previous experiments.

The findings in Fig. 2c clearly demonstrate the severe impact of neglecting guidance gradients in *Guided DBP* setups. Both the raw error (g_d) and the relative error (g_e) increase steeply with t^*, confirming that as the purification path grows longer, the omitted gradient paths—stemming from guidance dependencies—dominate the overall gradient signal. Most notably, g_e exceeds 10^4 at $t^* = 0.036$, indicating that the gradient used in prior works is essentially meaningless for optimization, as it no longer aligns with the true direction of steepest descent.

These findings directly explain the poor attack success rates previously observed on guided defenses like *GDMP* and *MimicDiffusion*, and reinforce why our attacks, which correctly account for these guidance gradients via **DiffGrad**, achieve drastically superior performance—see §4.2 and §6. In short, the guidance mechanism, when improperly handled, not only fails to help but actively hinders attack effectiveness by yielding incorrect gradient signals.

(d) Effect of incorrect surrogate gradients. As explained in §3.2 and Appendix D.1.5, prior work [41] uses the **surrogate** method [24] to enable efficient gradient computation for *DBP*. However, we identify a critical divergence in **DiffHammer**'s [41] implementation of this method that substantially compromises attack performance. This not only leads to inflated robustness estimates for *DBP*, but also to misleading conclusions about the relative effectiveness of the standard gradient-based attacks versus **DiffHammer**'s proposed enhancements—see §4.2. Specifically, as detailed in Appendix D.1.5, **DiffHammer** mismatches the time discretization used in the forward and backward passes: the forward pass computes intermediate denoised states $\hat{\boldsymbol{x}}(t)$ using a fine step size dt, while the backward pass attempts to propagate gradients at a coarser granularity $\bar{dt} > dt$ using those same (misaligned) forward states. This inconsistency leads to gradients being applied to states generated under a different dynamics, corrupting the computation and severely degrading the attack.

To validate this, we conduct a final experiment: For three different values of $dt \in \{0.001,\ 0.005,\ 0.01\}$ and fixed $t^* = 0.1$, we purify a fixed ***CIFAR-10*** sample $\boldsymbol{x}$. For each dt, we evaluate four surrogate variants using $\bar{dt} \in \{dt,\ 2dt,\ 5dt,\ 10dt\}$. For every configuration, we compute $\boldsymbol{g}_{nf}$ using **DiffHammer**'s surrogate implementation and $\boldsymbol{g}_f$ using the correct implementation. As before, we report both g_d and g_e to quantify the absolute and relative gradient error due to the mismatch between dt and $\bar{dt}$. Each experiment is repeated 10 times, and we report the mean values.

The results in Fig. 2d clearly validate our claim: the gradient errors induced by **DiffHammer**'s surrogate implementation grow substantially with increasing $\bar{dt}/dt$, saturating quickly even for modest mismatches. Both absolute error g_d and relative error g_e consistently worsen across all dt configurations, indicating that gradients are being applied to the wrong points in the computation graph.

Crucially, **DiffHammer** uses $dt = 0.01$ and $\bar{dt} = 2dt = 0.02$, a setup which already produces substantial degradation ($g_d \approx 3$, $g_e \approx 1$), confirming that their reported results rely on gradients that diverge significantly from the correct ones. These errors propagate into the attack, weakening its effectiveness and misleadingly suggesting that first-order attacks are inherently inferior. This experiment conclusively demonstrates that **DiffHammer**'s surrogate misuse is not a minor detail, but a critical bug that undermines their central claims.

We note that **DiffGrad** also provides a correct implementation of the **surrogate** method (despite the full correct gradient being the main method considered in the paper).

D.3 Pseudo-Code for Our Memory-Efficient Gradient-Enabled Purification with DiffGrad

Algorithm 1 Differentiable Purification with ***DiffGrad*** — Forward Propagation

Require: Sample $\boldsymbol{x}$, Score model $\boldsymbol{s}_\theta$, Optimal diffusion time t^*, step size dt, Noise scheduler β, Reverse diffusion function **calc_dx**, Noise sampler initializer **init_noise_sampler**, Guidance condition $\mathbf{g_{fn}}$, Guidance scale s, Auxiliary guidance extractor **g_aux**

1: $steps \leftarrow \left\lfloor \frac{t^*}{dt} \right\rfloor$, $\mathbf{guide} \leftarrow \mathbf{g\_aux}(\boldsymbol{x})$ /* Calc. #steps and init. guide */
2: **disable_dependencies**() /* Dependencies enabled during forward propagation. Disable them. */
3: $\mathbf{S} \leftarrow []$ /* Saved state (will eventually hold all intermediate reverse steps' outputs). */
4: $seed \leftarrow$ **random_seed**() /* Seed to initialize noise path */
5: $\mathbf{NS} \leftarrow$ **init_noise_sampler**($seed$) /* **Reproducible** sampler */
6: $\alpha \leftarrow$ **calc_alpha**(β) /* Calculate α factors from *eq. (2)* */
7: Draw $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I}_d)$
8: $\hat{\boldsymbol{x}} \leftarrow \sqrt{\alpha(t^*)}\boldsymbol{x} + \sqrt{1-\alpha(t^*)}\boldsymbol{\epsilon}$ /* Diffuse according to *eq. (2)* */
9: **for** $i \leftarrow steps, steps-1, ..., 1$ **do**
10: $\quad \mathbf{S}$.**append**($\hat{\boldsymbol{x}}$) /* Set $\mathbf{S}[i] = \hat{\boldsymbol{x}}(t)$. */
11: $\quad$ **step_noise** $\leftarrow \mathbf{NS}$.**sample**(i) /* Sample the random noise used to calculate $d\hat{\boldsymbol{x}}$ at step i. */
12: $\quad d\hat{\boldsymbol{x}} \leftarrow$ **calc_dx**($\hat{\boldsymbol{x}}, \boldsymbol{s}_\theta, i, dt, \beta,$ /* Calc. $d\hat{\boldsymbol{x}}$ according to
13: $\qquad$ **step_noise**, $\mathbf{g_{fn}}, s,$ **guide**) *eq. (4)* */
14: $\quad \hat{\boldsymbol{x}} \leftarrow \hat{\boldsymbol{x}} + d\hat{\boldsymbol{x}}$ /* Update $\hat{\boldsymbol{x}} = \hat{\boldsymbol{x}}(t + dt)$. */
15: **end for**
16: **enable_dependencies**() /* Re-enable dependencies. */
17: **return** $\hat{\boldsymbol{x}}, \mathbf{S}, \mathbf{NS},$ **guide**

Forward Propagation. **DiffGrad**'s forward propagation logic is in Algorithm 1. The code in blue is optional, pertaining to the use of guidance. We highlight in red the portions that differ from standard forward propagation. First, we generate the guidance $\boldsymbol{guide}$ from $\boldsymbol{x}$ (line 1) and disable all graph dependency storage (line 2), enabling our code to run efficiently without attempting to store graphs that will lead to memory failures. Afterward (lines 3-5), we initialize $\boldsymbol{S}$ as an empty list and

draw a random seed that is then used to invoke the abstract function $\boldsymbol{init\_noise\_sampler}$, which returns a noise sampler that provides a reproducible random path for the backpropagation phase (see §3.2). After the input is diffused (lines 6-8) via *eq. (2)*, lines 9-15 correspond to the reverse pass: At each step t (effectively i), $\hat{\boldsymbol{x}}$ (that now represents $\hat{\boldsymbol{x}}(t)$) is first appended to $\boldsymbol{S}$, which will eventually contain all such intermediate outputs (line 10). The noise provided by $\boldsymbol{NS}$ for the current step i is then retrieved (line 11) and used to compute $d\hat{\boldsymbol{x}}$ (line 12). $d\hat{\boldsymbol{x}}$ is then added to $\hat{\boldsymbol{x}}$ so that its current value becomes $\hat{\boldsymbol{x}}(t + dt)$. This repeats until $\hat{\boldsymbol{x}} = \hat{\boldsymbol{x}}(0)$. Unlike the naive implementation, we only store the intermediate results. For efficiency, we also avoid saving the random noise for each step i but utilize $\boldsymbol{NS}$ to reproduce those variables on demand. Before termination, we re-enable dependency storage (line 16) to ensure our code does not interfere with the execution of any other modules. Finally, $\hat{\boldsymbol{x}}(0)$ is returned together with the state $\boldsymbol{S}$ and the sampler $\boldsymbol{NS}$, which are stored internally for reproducibility during backpropagation.

DBP Parallelism. Although Algorithm 1 is presented for a single purification path, **DiffGrad** natively supports parallel purification of N stochastic copies to potentially obtain a significant speedup when computing higher-quality EOT gradients—see §3.2. For simplicity, we abstracted away the batch logic in the pseudo-code. In practice, prior to line 1, the input $\boldsymbol{x}$ is replicated N times (N is an additional argument that can be provided to Algorithm 1). Line 4 returns N random seeds, and the resulting sampler $\mathbf{NS}$ manages N reproducible random paths. Line 7 draws N distinct noise samples to generate N diffused versions of $\boldsymbol{x}$, and all subsequent steps operate copy-wise in parallel across these N instances.

Backpropagation. **DiffGrad**'s backpropagation logic is in Algorithm 2. Similar to earlier, red text refers to operations that deviate from traditional backpropagation, while blue lines are optional (guidance-related). In addition to the usual gradient $\boldsymbol{grad}$ w.r.t. $\hat{\boldsymbol{x}}(0)$, the inputs include multiple parameters normally exclusive to forward propagation, as they are required to re-calculate the dependencies. Additionally, the algorithm accepts the saved state $\boldsymbol{S}$, and the same noise sampler $\boldsymbol{NS}$ to retrieve the stochastic path of the forward propagation. Before providing details, we note that by definition $\forall \boldsymbol{A}, \boldsymbol{B} \in \mathbb{R}^d$, it holds that:

$$\langle \boldsymbol{A}, \boldsymbol{B} \rangle = \sum_d \boldsymbol{A} \odot \boldsymbol{B}$$

where $\odot$ denotes the element-wise product. Therefore, in order to calculate the gradients w.r.t. $\hat{\boldsymbol{x}}(t)$ and $\boldsymbol{guide}$ as described in *eq. (6)* and *eq. (9)*, we may define an objective at each step t as:

$$Obj_t = \sum_d (\hat{\boldsymbol{x}}(t + dt) \odot \nabla_{\hat{\boldsymbol{x}}(t+dt)} F) \tag{12}$$

and take its gradient w.r.t. the two elements of interest above, which explains the steps in our pseudo-code in Algorithm 2.

The procedure begins by creating a variable $\boldsymbol{g\_grad}$ and setting it to $\mathbf{0}$ (line 2). This will later be used to store the guidance gradients (see Appendix D.1.4). For each time step t (i.e., step i), starting from $t' = -dt$ ($i = 1$), the process (lines 3-14) first retrieves $\hat{\boldsymbol{x}}(t)$ from the saved state S (line 4) and the corresponding random noise for that step used during forward propagation (line 5) and computes $\hat{\boldsymbol{x}}(t + dt)$, denoted as $\hat{\boldsymbol{x}}_{+dt}$ (lines 7-9). Importantly, these computations are performed while storing graph dependencies (enabled on line 6 and re-disabled on line 11 to restore the normal execution state). Specifically, during the first step, we calculate $\hat{\boldsymbol{x}}(0)$ from $\hat{\boldsymbol{x}}(-dt)$. Afterward, we compute the objective Obj_t (line 10) following *eq. (12)* that allows us to back-propagate the gradient from $\hat{\boldsymbol{x}}(0)$ to $\hat{\boldsymbol{x}}(-dt)$ and $\boldsymbol{guide}$ using the stored dependencies, as per *eq. (6)* and *eq. (9)*. $\boldsymbol{grad}$ is then updated to hold the gradient of the loss function w.r.t. $\hat{\boldsymbol{x}}(-dt)$ as desired (line 12), and the gradient of $\boldsymbol{guide}$ due to this guidance path (i.e., from $\boldsymbol{guide}$ to the loss due to $\boldsymbol{guide}$ participating directly in the calculation of $\hat{\boldsymbol{x}}(t + dt)$— see Appendix D.1.4) is added to $\boldsymbol{g\_grad}$ (line 13). This process repeats until $\boldsymbol{grad}$ finally holds the gradient w.r.t. $\hat{\boldsymbol{x}}(t^*)$ and $\boldsymbol{g\_grad}$ holds the sum of gradients due to all guidance paths w.r.t. $\boldsymbol{guide}$ (*eq. (10)*). Note that after the required gradients w.r.t. $\hat{\boldsymbol{x}}(t)$ and $\boldsymbol{guide}$ are obtained at each step, the dependencies are no longer needed and can be discarded. This is where our approach differs from traditional backpropagation algorithms, enabling memory-efficient gradient calculations (at the cost of an additional forward propagation in total). At this point (line 14), we have the gradient $\nabla_{\hat{\boldsymbol{x}}(t^*)} F$ and all is required is to use it to calculate $\nabla_{\boldsymbol{x}} F$, which is trivial

Algorithm 2 Differentiable Purification with ***DiffGrad*** — Backpropagation

Require: Loss gradient $\boldsymbol{grad}$ w.r.t $\hat{\boldsymbol{x}}_0$, Sample $\boldsymbol{x}$, Score model $\boldsymbol{s}_\theta$, Optimal diffusion time t^*, step size dt, Noise scheduler β, Reverse diffusion function $\mathbf{calc\_dx}$, State $\mathbf{S} = \{\hat{\boldsymbol{x}}(dt{*}i) | i \in [\![|\frac{t^*}{dt}|]\!]\}$, Noise sampler $\mathbf{NS}$, Auxiliary guidance input $\mathbf{guide}$, Guidance function $\mathbf{g_{fn}}$, Guidance scale s

1: $steps \leftarrow \left|\frac{t^*}{dt}\right|$
2: $\mathbf{g\_grad} \leftarrow \mathbf{0}$ /* Init. gradient w.r.t guidance input */
3: **for** $i \leftarrow 1, 2, ..., steps$ **do**
4: $\quad \hat{\boldsymbol{x}} \leftarrow \mathbf{S}[i]$ /* Set $\hat{\boldsymbol{x}} = \hat{\boldsymbol{x}}(t)$ */
5: $\quad \mathbf{step\_noise} \leftarrow \mathbf{NS.sample}(i)$ /* Retrieve noise for step i */
6: $\quad \mathbf{enable\_dependencies}()$
7: $\quad d\hat{\boldsymbol{x}} \leftarrow \mathbf{calc\_dx}(\hat{\boldsymbol{x}}, \boldsymbol{s}_\theta, i, dt, \beta,$
8: $\qquad \mathbf{step\_noise}, \mathbf{g_{fn}}, s, \mathbf{guide})$
9: $\quad \hat{\boldsymbol{x}}_{+dt} \leftarrow \hat{\boldsymbol{x}} + d\hat{\boldsymbol{x}}$ /* $\hat{\boldsymbol{x}}_{+dt} = \hat{\boldsymbol{x}}(t + dt)$ */
10: $\quad Obj_t \leftarrow \sum(\hat{\boldsymbol{x}}_{+dt} \odot \boldsymbol{grad})$ /* Objective due to *eq. (12)* */
11: $\quad \mathbf{disable\_dependencies}()$
12: $\quad \boldsymbol{grad} \leftarrow \nabla_{\hat{\boldsymbol{x}}} Obj_t$ /* Update gradient w.r.t $\hat{\boldsymbol{x}}(t)$ (*eq. (6)*) */
13: $\quad \mathbf{g\_grad} \leftarrow \mathbf{g\_grad} + \nabla_{\mathbf{guide}} Obj_t$ /* Update **guide** gradient (*eq. (10)*) */
14: **end for**
15: $\alpha \leftarrow \mathbf{calc\_alpha}(\beta)$
16: $\boldsymbol{grad} \leftarrow \boldsymbol{grad} * \sqrt{\alpha(t^*)}$ /* Loss gradient w.r.t $\boldsymbol{x}$ (*eq. (2)*) */
17: $\mathbf{g\_grad} \leftarrow \nabla_{\boldsymbol{x}} \sum(\mathbf{guide} \odot \mathbf{g\_grad})$ /* Guidance gradient w.r.t $\boldsymbol{x}$ (*eq. (11)*) */
18: $\boldsymbol{grad} \leftarrow \boldsymbol{grad} + \mathbf{g\_grad}$ /* Merge loss and guidance gradients */
19: **return** $\boldsymbol{grad}$

due to the chain rule since the closed-form solution for $\hat{\boldsymbol{x}}(t^*) \equiv \boldsymbol{x}(t^*)$ from *eq. (2)* indicates that this is equivalent to $\nabla_{\boldsymbol{x}} F = \sqrt{\alpha(t^*)} * (\nabla_{\boldsymbol{x}(t^*)} F)$ as we compute on line 16. We then calculate the guidance paths' gradient w.r.t $\boldsymbol{x}$ following *eq. (11)* on line 17. Finally, we sum both components, returning the precise full gradient w.r.t. $\boldsymbol{x}$.

DBP Parallelism. Algorithm 2 is written for a single purified copy but in practice operates over a batch of N stochastic instances like Algorithm 1. As the forward pass stores N trajectories, the backward pass propagates gradients independently for each. The operations in the pseudo-code are thus applied copy-wise in parallel across all N instances. Finally, after line 18, the N gradients are averaged, yielding the EOT gradient.

D.4 Verifying the Correctness of DiffGrad

To ensure the correctness of **DiffGrad**, all that is required is to verify that each $\hat{\boldsymbol{x}}(t + dt)$ computed during backpropagation (line 9 in Algorithm 2) exactly matches the corresponding forward-computed $\hat{\boldsymbol{x}}(t + dt)$ (line 14 in Algorithm 1). This equality guarantees that the reconstructed computation graph faithfully mirrors the one produced by standard autodiff engines during their normal (non-checkpointed) operation, ensuring exact gradient recovery since we use them to perform the necessary backpropagation between each $\hat{\boldsymbol{x}}(t + dt)$ and $\hat{\boldsymbol{x}}(t)$.

Since our forward pass explicitly stores all intermediate states in $\boldsymbol{S}$, we compare each recomputed $\hat{\boldsymbol{x}}(t + dt)$ with its forward counterpart during backpropagation. An exact match confirms that the system is computing precise gradients. We manually validated this for all timesteps.

For guidance gradients, correctness follows from the derivations in Appendix D.1.4. Once incorporated, the same matching procedure ensures their correctness as well.

E Additional Details on Systems & Models

WideResNet-28-10 and WideResNet-70-16 [52] are used for ***CIFAR-10***, and ResNet-50 [18], WideResNet-50-2, and DeiT-S [14] for ***ImageNet***, similar to [30, 20]. For **VP-SDE** *DBP* (*DiffPure*) [30], the *DM*s [12, 36] are those from the original work. We also experiment with the *Guided*-**DDPM** (see §3.2), *GDMP* [40], due to its *SOTA* robustness, using the author-evaluated

*DM*s [12, 19]. The settings match the original optimal setup [40, 30]: For *Diffpure*, $t^*=0.1$ for ***CIFAR-10*** and $t^*=0.15$ for ***ImageNet***. For *GDMP*, a ***CIFAR-10*** sample is purified $m=4$ times, with each iteration running for 36 steps ($t^*=0.036$), using *MSE* guidance [40]. ***ImageNet*** uses 45 steps ($m=1$) under **DDPM**-acceleration [12] with *SSIM* guidance [43].

F One-Shot *DBP* Baseline Comparisons Against DiffGrad's Accurate Gradients

As noted in §3, *DBP*'s robustness stems from two sources: inaccurate gradients and improper evaluation. As our work offers enhancements on both fronts, we evaluate each factor separately. Here, we isolate the gradient issue by re-running prior experiments under the same one-shot evaluation protocol, but with accurate gradients via our **DiffGrad** module and compare the results to those from the literature. We restrict our attacks to 10 optimization steps (with *AA*-ℓ_∞), while prior works often use up to 100, giving them a clear advantage. We evaluate on ***CIFAR-10*** with WideResNet-28-10, using $N = 128$ EOT samples as justified in §3.2. For papers reporting several results, we chose their best.

Baselines. To demonstrate the efficacy of exact full gradients, we compare our **DiffGrad** module against prior gradient approaches. For *DiffPure*, the **adjoint** method was originally used [30], while *GDMP* was initially evaluated using **BPDA** [2] and a **blind** variant (ignoring the defense entirely) in Wang et al. [40]. These approximations were later criticized for poor attack performance [41, 26, 20, 24]. More recently, Lee and Kim [24] proposed the **surrogate** process to approximate the gradients, performing the reverse pass with fewer steps during backpropagation to reduce memory usage, enabling approximate gradients via standard autodiff tools. **DiffAttack** [20], **DiffHammer** [41], and Liu et al. [26] proposed checkpointing for memory-efficient full-gradient backpropagation. However, **DiffHammer** avoids the standard one-shot evaluation (1-evaluation) protocol and reports no results under it, while **DiffAttack** does not evaluate *GDMP* and continues to use the **adjoint** method for *DiffPure*. We focus here on existing results under the 1-evaluation protocol and defer the discussion of these works' conceptual issues (see §1) to §4.2.

Metric. Following prior works, we report *robust accuracy* (*Rob-Acc*): the fraction of samples correctly classified after the attack completes and the final adversarial example is purified and evaluated (once).

Results. Table 4 shows our comparison. All methods achieve similar clean accuracy (Cl-Acc) without attacks, with minor variation due to sample selection. Thus, robust accuracy (Rob-Acc) differences reflect the effect of gradient methods. **Full** denotes the standard AA attack that uses the full exact gradients (i.e., via checkpointing).

Table 4: One-shot *AA*-ℓ_∞ comparison on ***CIFAR-10*** ($\epsilon_\infty=8/255$). † indicates strategy is *PGD*.

Models	Pur.	Gradient Method	Cl-Acc %	Rob-Acc %
WideResNet-28-10	*DiffPure* [30]	**Adjoint** (Nie et al. [30])	89.02	70.64
		DiffAttack (Kang et al. [20])	89.02	46.88
		Surrogate (Lee and Kim [24])†	90.07	48.28
		Full (Liu et al. [26])	89.26	62.11
		Full-DiffGrad (Ours)	89.46	**48.05**
	GDMP [40]	**Blind** (Wang et al. [40])	93.50	90.06
		BPDA (Lee and Kim [24])	89.96	75.59
		Surrogate (Lee and Kim [24])†	89.96	24.53
		Full-DiffGrad (Ours)	93.36	**19.53**

Our approach significantly outperforms gradient approximations such as **Adjoint**, **Blind**, and **BPDA**, reaffirming their known weaknesses. More notably, despite using only 10 optimization steps, our method reduces *DiffPure*'s Rob-Acc by 14.06% compared to Liu et al. [26], who also use exact gradients, confirming their backpropagation mismatches (see §3.2).

While **DiffAttack** remains slightly stronger, the difference is extremely small (1.4%) and can be safely attributed to differing evaluation samples. Importantly, the original gap reported in Kang et al. [20] between **DiffAttack** and the standard *AA*-ℓ_∞ dropped by 23.76%, undermining its claimed advantage due to the per-step deviated reconstruction losses and aligning with our theoretical findings in §3.1. As **DiffAttack** involves broader architectural changes beyond gradient logic, we provide detailed comparisons in §4.2, where we clearly showcase its inferiority.

Our method also slightly outperforms **Surrogate** for *DiffPure* (0.23%), but we caution against overinterpreting this small gap: under the improper one-shot evaluation, several purification paths can

still cause misclassification even if the majority yield correct labels. As such, good approximation methods like **Surrogate** may appear closer in performance than they truly are. To verify the superiority of our **DiffGrad**'s full gradients over the **surrogate** approximation against *DiffPure*, we thus further compare the two under the realistic *MV* protocol studied in §4.2, executing *AA*-ℓ_∞ with the **surrogate** process to obtain the gradients when attacking the same ***CIFAR-10*** classifier considered in §4.2 (i.e., WideResNet-70-16) and using the same number of samples ($N = 10$) over which the majority vote is taken. We find that the **surrogate** process brings MV.Rob to 43.75% only, whereas our **Full-DiffGrad** lowers this number to 39.45%, achieving a considerable improvement of 4.3% and unequivocally proving the advantage of exact gradient computations. Finally, for *GDMP*, our method outperforms **Surrogate** by 5% even in the one-shot evaluation protocol (see Table 4), largely due to our incorporation of guidance gradients—entirely absent in all prior approaches—highlighting the unique strength of **DiffGrad**.

Note that the **surrogate** method shortens the trajectory and backpropagates through this shorter horizon but retains the full graph over that path. Hence, one may argue that the attack success degradation is an acceptable tradeoff when using the **surrogate** method given the potential speedup. Yet, in *DBP*, memory—not FLOPs—is the bottleneck; Checkpointing (e.g., our **DiffGrad**) recomputes individual path segments, yielding exact gradients with lower peak memory. The **surrogate** may improve speed but not memory pressure, and its truncation can miss dependencies, weakening gradients as shown above. In practice, attackers precompute adversarial examples offline since *DBP*'s latency makes real-time adversarial generation infeasible either way, so runtime savings alone do not translate to a practical advantage.

All in all, the results demonstrate the fragility of *DBP* in the face of accurate gradients and highlight the issues in previous works' backpropagation. Nonetheless, this one-shot evaluation protocol remains problematic, as previously stated, leading to an inflated robustness estimate. In fact, under more realistic settings (see §4.2), we demonstrate that this gradient-based attack almost entirely defeats *DBP* when only one sample is used to predict the label (i.e., Wor.Rob).

G Evaluations with Liu et al. [26]'s Fixed *AutoAttack*

Liu et al. [26], like **DiffHammer** [41] and our own analysis, note that evaluating a single purification at attack termination inflates robustness scores. Liu et al. [26] address this by evaluating 20 replicas of the final *AE*, declaring success if any is misclassified. While similar in spirit to the Wor.Rob metric we consider (see §3.3), this protocol is more limited: it evaluates only at the final step, whereas Wor.Rob evaluates N copies at each attack iteration. Accordingly, Liu et al. [26] group their method with one-shot evaluations, providing a slightly more realistic assessment that leads to results similar to those attained via *PGD* [23].

Table 5: *AA*-ℓ_∞ comparison on ***CIFAR-10*** **under Liu et al. [26]'s protocol (i.e., Fixed *AutoAttack*)** (ϵ_∞=$8/255$).

Models	Pur.	Gradient Method	Cl-Acc %	Rob-Acc %
WideResNet-28-10	*DiffPure* [30]	**Full** (Liu et al. [26])	89.26	56.25
		Full-DiffGrad	89.46	**30.86**
	GDMP [40]	**Full** (Liu et al. [26])	91.80	40.97
		Full-DiffGrad	93.36	**10.55**
WideResNet-70-16	*DiffPure* [30]	**Full-DiffGrad**	89.06	**35.16**
	GDMP [40]	**Full-DiffGrad**	91.8	**8.59**

We replicate their setup using **DiffGrad** (Table 5), running 20 iterations with $N = 10$ EOT samples per step and evaluating 20 final replicas. Under this protocol, our improvements are stark. On WideResNet-28-10, we reduce Rob-Acc by 25.39% and 30.42% for *DiffPure* and *GDMP*, bringing final accuracy to 30.86% and 10.55%, respectively. These results confirm the superiority of **DiffGrad**'s gradients and expose *DBP*'s realistic vulnerability. We observe similar results on WideResNet-70-16 (not evaluated in [26]).

H Ablation Study with Different Numbers of Samples for Label Prediction

To assess the impact of the number of evaluation samples N, we test both single-purification and majority-vote (MV) settings with $N \in \{1, 10, 128\}$. Our goal is to highlight the brittleness of *DBP*'s standard deployment, which classifies based on a single purified copy. As explained in §3.3, in

stateless setups (e.g., phishing, spam, CSAM), adversaries can resubmit identical queries indefinitely. Since randomized defenses like *DBP* can fail along certain stochastic paths, any non-negligible misclassification probability compounds with repeated attempts—see §3.3. Although *DBP* assumes this probability is negligible, our proof in §3.1, results in §4, and the ablation experiments here contradict this. Furthermore, we wish to showcase the advantages of our proposed majority-vote *(MV)* setting that strives to mitigate this vulnerability by, instead, classifying based on expectation of the randomized defense.

We evaluate *DiffPure* and *GDMP* on ***CIFAR-10*** using WideResNet-70-16 and (our) *AA*-ℓ_∞ (with $\epsilon = 8/255$ and 100 iterations). For $N = 1$ and $N = 10$, we use 10 EOT samples; for $N = 128$, we reuse the same 128 samples for both EOT and evaluation. This improves gradient quality, thus strengthening attacks, yet still leads to higher *MV* robustness, proving the superiority of our proposed method despite the use of more accurate gradients.

For Wor.Rob, clean accuracy (Cl-Acc) remains constant across all N values since we always compute it using a single purified copy ($N = 1$), independent of the number N of samples used in the attack. This reflects the actual standard single-purification deployment of *DBP*, where the defender classifies a single output, while the attacker may retry multiple times. Hence, for Rob-Acc we consider a batch of multiple samples (the corresponding N in that row) and then declare attack success if a single misclassification occurs, revealing the gap between measured and effective robustness. In contrast, MV.Rob clean accuracy varies with N, as the prediction always aggregates over N purified samples, consistent with our proposed deployment.

Table 6 confirms that Wor.Rob consistently declines with N, revealing the illusion of robustness under single evaluation. For instance, *DiffPure* shows a drop from 35.16% (at $N = 1$) to 17.58% at $N = 128$ (far lower than the inflated numbers reported in previous studies—see Appendix F), confirming that many stochastic paths yield incorrect predictions, as our properly implemented gradient-based attack (see §3.2) lowers the expected classification confidence, making such failure modes more likely. For *GDMP*, we observe a similar trend.

Table 6: *AA*-ℓ_∞ Performance under various evaluation sample counts

Pur.	#Samples	Wor.Rob %		MV.Rob %	
		Cl-Acc	Rob-Acc	Cl-Acc	Rob-Acc
DiffPure [30]	**N=1**	89.06	35.16	89.06	35.16
	N=10		17.19	91.02	39.45
	N=128		17.58	92.19	47.72
GDMP [40]	**N=1**	91.8	8.59	91.8	8.59
	N=10		7.03	92.19	16.8
	N=128		5.47	92.19	32.81

In contrast, MV.Rob improves with N, rising from 35.16% to 47.72% for *DiffPure*, and from 8.59% to 32.81% for *GDMP*. This affirms *MV* as a more stable and accurate evaluation method that must be adopted as the de facto standard for *DBP* evaluations in the future.

Yet, despite *MV*'s benefits, our gradient-based *AA*-ℓ_∞ attack still degrades robustness, which never exceeds 50%, validating our theoretical finding from §3.1 that such attacks repurpose *DBP* into an adversarial distribution generator.

Computational Cost. While larger N increases computational overhead, we identify $N = 128$ as the max batch size fitting in a single A100 GPU run requiring $\sim 26.53s$ for inference, with $N = 10$ offering a practical middle ground ($6.54s$ vs. $5.29s$ for $N = 1$). Hence, for practicality and if throughput is critical, one should opt for $N = 10$. However, in security-critical systems where latency is not crucial, a larger N is favorable. Yet, additional tests (not shown) suggest MV.Rob plateaus near $N = 128$. Note that these latencies refer to purification inference alone (classification excluded). During attacks (not standard deployment), there is also the cost of backpropagation. When accounting for the cost of classification and backpropagation, the latency becomes $\sim$ 17s for $N = 1$ and $N = 10$ compared to $\sim$ 53s for $N = 128$. As explained in §3.2, our design allows purifying multiple stochastic copies of the same sample in one step, in contrast to previous work. This reduces the runtime per EOT gradient from up to $N \times$ 17s—where N stochastic purifications are run serially—to just T_Ns, where T_N is the latency incurred by purifying N samples in a batch, yielding a speedup of up to $N \times 17/T_N$ for single-sample attacks, which amounts to $41.06\times$ for $N = 128$ allowing for a considerable speedup when the objective is to utilize many EOT samples to obtain accurate gradient estimates. For batch attacks, prior methods purify one stochastic copy per sample per EOT step and require all samples to converge before proceeding to the next batch of

inputs, blocking early termination. In contrast, **DiffGrad** purifies N copies of a single sample in parallel, allowing samples to terminate independently. This greatly improves overall throughput by freeing compute sooner, especially when convergence varies across inputs.

ImageNet** timings:* While we limit our ablation studies to ***CIFAR-10 for feasibility, we provide ***ImageNet*** timings for completeness. At ***ImageNet*** resolution on a 40 GB A100 GPU, *DBP* inference (forward propagation only) requires $\sim$ 7s for $N = 1$ and $\sim$ 25s for $N = 8$. A single attack iteration (i.e., forward and backward propagation) lasts $\sim$ 16s ($N = 1$) and $\sim$ 75s ($N = 8$). Thus, batching EOT copies yields a significant speedup over serial purification for both attacks and inference, but models and activations dominate cost at this scale, so batching returns diminish faster than on ***CIFAR-10***.

Recommendations for Choosing The Batch Size N. We found $N = 128$ for ***CIFAR-10*** and $N = 8$ for ***ImageNet*** to be the largest feasible batch sizes to fit into the 40 GB A100 GPU memory, making larger batches impractical for large-scale evaluations. That said, control experiments with larger N values for both datasets (split into batches) showed negligible robustness or attack performance gains, indicating saturation.

Following the above discussion, we recommend $N = 10$ for ***CIFAR-10*** as a practical default and $N = 64 - 128$ for security-critical inference (*MV*) and highly accurate gradients when evaluating new attack methods such as our *LF* strategy—see §5. For ***ImageNet***, we recommend $N = 8$ for similar reasons; larger N values remain preferable when added latency is acceptable.

I Details on Our Low-Frequency (*LF*) Adversarial Optimization Strategy

I.1 Understanding Optimizable Filters

In practice, *OFs* extend an advanced class of filters, namely ***guided*** filters, that improve upon the basic filters discussed in §5. ***Guided*** filters employ additional per-pixel *color kernels* that modulate the distortion at critical points: Since filters interpolate each pixel with its neighbors, they are destructive at edges (intersections between different objects in the image), while the values of non-edge pixels are similar to their neighbors, making such operations of little effect on them. Depending on a permissiveness σ, ***guided*** filters construct, **for each pixel** $(i,\ j)$ a *color kernel* $\boldsymbol{c}^{i,j}_{\boldsymbol{x},\sigma_c}$ of the same dimensionality $M{\times}N$ as $\boldsymbol{\mathcal{K}}$ that assigns a multiplier for each of $(i,\ j)$'s neighbors, that decays with the neighbor's difference in value from $(i,\ j)$'s. The output at $(i,\ j)$ involves calculating the effective kernel $\boldsymbol{\nu}^{i,j}_{\boldsymbol{x},\boldsymbol{\mathcal{K}},\sigma_c} = \boldsymbol{c}^{i,j}_{\boldsymbol{x},\sigma_c} \odot \boldsymbol{\kappa}$ (normalized) which then multiplies $(i,\ j)$'s vicinity, taking the sum of this product. Thus, contributions from neighbors whose values differ significantly are diminished, better preserving information.

Guided filters still employ the same $\boldsymbol{\mathcal{K}}$ for all pixels, changing only the *color kernel* that is computed similarly for all pixels. Thus, to incur sufficient changes, they would also require destructive parameters despite them still potentially performing better compared to their pristine counterparts. Their parameters are also predetermined, making it impossible to optimize them for a specific purpose. The *OFs* by Kassis and Hengartner [21] build upon ***guided*** filters but differ in two ways: First, instead of using the same $\boldsymbol{\mathcal{K}}$, they allow each pixel to have its own kernel $\boldsymbol{\mathcal{K}}^{i,j}$ to better control the filtering effects at each point, ensuring visual constraints are enforced based on each pixel's visual importance. In this setting, $\boldsymbol{\mathcal{K}}^*$ denotes the set including all the per-pixel kernels $\boldsymbol{\mathcal{K}}^{i,j}$. Second, the parameters $\boldsymbol{\theta}_{\boldsymbol{\mathcal{K}}^*}$ of each filter are learnable using feedback from a perceptual metric (***lpips***) [56] that models the human vision, leading to an optimal assignment that ensures similarity while maximizing the destruction at visually non-critical regions. To further guarantee visual similarity, they also include *color kernels* similar to ***guided*** filters (see original paper for details [21]).

I.2 Attack Hyperparameters

Through experimentation, we found that the loss balancing constants $c = 10^8$ for ***CIFAR-10*** and $c = 10^4$ for ***ImageNet*** lead to the fastest convergence rates and selected these values accordingly (although other choices are also possible). ***UnMarker***'s filter network architecture for ***ImageNet*** is identical to that from the original paper [21]. For ***CIFAR-10***, since the images are much smaller, we found the original architecture unnecessarily costly and often prevents convergence since larger filters group pixels from distant regions together in this case, easily violating visual constraints upon each

update, resulting in the ***lpips*** condition being violated. Thus, we opt for a more compact network that includes filters with smaller dimensions, which was chosen based on similar considerations to [21], allowing us to explore several interpolation options. The chosen architecture for ***CIFAR-10*** includes 4 filters, whose dimensions are: $(5,5),\ (7,7),\ (5,5),\ (3,3)$. We use fixed learning rates of 0.008 for the direct modifier $\boldsymbol{\delta}$ and 0.05 for the filters' weights, optimized using $\boldsymbol{Adam}$. The remaining hyperparameters were left unchanged compared to [21].

I.3 Pseudo-Code

The pseudo-code for our low-frequency (*LF*) strategy (see §5) is in Algorithm 3. Importantly, as each $\boldsymbol{\mathcal{K}}_b^{i,j}$'s values should be non-negative and sum to 1, the values for each such per-pixel kernel are effectively obtained by ***softmax***ing the learned weights. Initially, the modifier $\hat{\boldsymbol{\delta}}$ is initialized to $\mathbf{0}$ and the weights $\{\widehat{\boldsymbol{\theta}}_{\boldsymbol{\mathcal{K}}_b^*}\}$ are selected s.t. the filters perform the identity function (line 1). As a result, the attack starts with $\boldsymbol{x}_{adv} = \boldsymbol{x}$ that is iteratively optimized. Similar to *C&W* [4], we directly optimize $\boldsymbol{x}_{adv}$ (i.e., using the modifier $\hat{\boldsymbol{\delta}}$) in the $\boldsymbol{arctanh}$ space, meaning we transform the sample first to this space by applying $\boldsymbol{arctanh}$ (after scaling $\boldsymbol{x}_{adv}$ to $\boldsymbol{arctanh}$'s valid range $[-1,\ 1]$) where $\hat{\boldsymbol{\delta}}$ is added and then restore the outcome to the original problem space (i.e., $[min\_val,\ max\_val]$, which is typically $[0,\ 1]$) via the $\boldsymbol{tanh}$ operation. Further details on this method and its benefits can be found in [4]. All other steps correspond to the description brought in §5. Unlike §3.1, we assume the classifier $\mathcal{M}$ outputs the logit vector rather than the probabilities (i.e., we omit the $\boldsymbol{softmax}$ layer over its output, which me or may not be re-introduced by the loss ℓ), as is traditionally done for a variety of adversarial optimization strategies (e.g., *C&W* [4]) to avoid gradient vanishing. We use the known ***max-margin*** loss [4]— $\ell(\boldsymbol{logits}, y) = \boldsymbol{logits}[:y] - \max\limits_{j \neq y} \{\boldsymbol{logits}[:j]\}$.

Algorithm 3 Low-Frequency (*LF*) Adversarial Optimization

Require: Sample $\boldsymbol{x}$, Model (classifier) $\mathcal{M}$, *DBP* pipeline D, Loss function ℓ, True label y of $\boldsymbol{x}$, Perceptual loss $\boldsymbol{lpips}$, $\boldsymbol{lpips}$ threshold τ_p, Filter architecture $\prod_{\boldsymbol{OF}}^{B}$, Balancing constant c, Iterations max_iters, Success condition $\boldsymbol{Cond}$, Filter weights learning rate $lr_{\boldsymbol{OF}}$, Modifier learning rate $lr_{\boldsymbol{\delta}}$, Number of purified copies n, Number of ***EoT*** eot_steps, Input range limits (min_val, max_val)

1: $\{\widehat{\boldsymbol{\theta}}_{\boldsymbol{\mathcal{K}}_b^*}\} \leftarrow \boldsymbol{identity\_weights}(\prod_{\boldsymbol{OF}}^{B}),\ \hat{\boldsymbol{\delta}} \leftarrow \mathbf{0}$ /* Initialize attack parameters. */
2: $\boldsymbol{Optim} \leftarrow \boldsymbol{Adam}([\{\widehat{\boldsymbol{\theta}}_{\boldsymbol{\mathcal{K}}_b^*}\}, \hat{\boldsymbol{\delta}}], [lr_{\boldsymbol{OF}}, lr_{\boldsymbol{\delta}}])$
3: $\boldsymbol{x}_{inv} \leftarrow \boldsymbol{inv\_scale\_and\_arctanh}(\boldsymbol{x}, min\_val, max\_val)$ /* Scale $\boldsymbol{x}$ to $[-1, 1]$ and take $\boldsymbol{arctanh}$ */
4: **for** $i \leftarrow 1$ **to** $max\_iters \cdot eot\_steps$ **do**
5: $\quad \boldsymbol{x}_{adv} \leftarrow \prod_{\boldsymbol{OF}}^{B}(\boldsymbol{tanh\_and\_scale}(\boldsymbol{x}_{inv} + \hat{\boldsymbol{\delta}}, min\_val, max\_val))$ /* Generate adversarial input using new $\{\widehat{\boldsymbol{\theta}}_{\boldsymbol{\mathcal{K}}_b^*}\}$ and $\hat{\boldsymbol{\delta}}$ via (*eq. (7)*) scaled to $[min\_val, max\_val]$. */
6: $\quad dist \leftarrow \boldsymbol{lpips}(\boldsymbol{x}, \boldsymbol{x}_{adv})$ /* Calculate perceptual distance. */
7: $\quad \hat{\boldsymbol{x}}_{adv}^0 \leftarrow D(\boldsymbol{repeat}(\boldsymbol{x}_{adv}, n))$ /* Get purified outputs. */
8: $\quad \boldsymbol{logits} \leftarrow \mathcal{M}(\hat{\boldsymbol{x}}_{adv}^0)$ /* Compute model output. */
9: $\quad$ **if** $\boldsymbol{Cond}(\boldsymbol{logits}, y)$ **and** $dist \leqslant \tau_p$ **then**
10: $\quad\quad$ **return** $\boldsymbol{x_{adv}}$ /* Success. Return $\boldsymbol{x}_{adv}$. */
11: $\quad$ **end if**
12: $\quad Objective \leftarrow \ell(\boldsymbol{logits}, y) + c \cdot max(dist - \tau_p, 0)$ /* Compute loss (*eq. (7)*). */
13: $\quad Objective.\boldsymbol{backward}()$ /* Get gradients for parameters. */
14: $\quad$ **if** $i \bmod eot\_steps = 0$ **then**
15: $\quad\quad \boldsymbol{Optim.step}()$ /* Update parameters. */
16: $\quad\quad \boldsymbol{Optim.zero\_grad}()$ /* Reset gradients. */
17: $\quad$ **end if**
18: **end for**
19: **return** $\boldsymbol{x}$ /* Failure. Return original $\boldsymbol{x}$. */

To average the gradients over multiple (N) paths as per the adaptive attack's requirements from §3.1, we generate several purified copies by repeating the sample $\boldsymbol{x}_{adv}$ under optimization n times before feeding it into the *DBP* pipeline (line 9). Here, n corresponds to the maximum number of copies we can fit into the *GPU*'s memory during a single run. However, as this n may be smaller than the desired N from §3.1 (i.e., number of *EoT* samples) that allows us to sufficiently eliminate the error in the computed gradient, we use gradient accumulation by only making updates to the optimizable parameters (and then resetting their gradients) every eot_steps iterations (lines 16-19). By doing so, the effective number of used copies becomes $n * eot\_steps$, which can represent any N of choice that is divisible by n. Note that if n is not a divisor of N, we can always increase N until this condition is met, as a larger N can only enhance the accuracy). This also explains why the algorithm runs for $max\_iters * eot\_steps$ (line 5).

Finally, the condition $\boldsymbol{Cond}$ captures the threat model (either *SP* or *MV*—see §2): When the logits for the batch of n copies are available together with the target label y, $\boldsymbol{Cond}$ outputs a success decision based on whether we seek misclassification for the majority of these purified copies or a single copy only. Note that, as explained in §4.2, we take the majority vote over the maximum number of copies we can fit into the *GPU* (i.e., n) for *MV*. As this choice was only made for practical considerations, one may desire to experiment with different configurations wherein another number of copies is used. Yet, this is easily achievable by simply modifying $\boldsymbol{Cond}$ accordingly: For instance, we may augment it with a history that saves the output logits over all eot_steps (during which $\boldsymbol{x}_{adv}$ is not updated). Then, the majority vote can be taken over all copies in this window. Note that by increasing the number of eot_steps, we can use as many copies for the majority vote decision as desired in this case. That said, the attack will become significantly slower.

In addition to the precise gradient module **DiffGrad**, our *DiffBreak* toolkit provides the implementation of our *LF* strategy as well as various other common methods (e.g., *AA* and *StAdv*), to enable robust and reliable evaluations of *DBP*. All strategies are optimized for performance to speed up computations via various techniques such as just-in-time (*JIT*) compilation. Our code is available at `https://github.com/andrekassis/DiffBreak`.

J Example Attack Images

In Appendix J.1-J.10, we provide a variety of successful attack images that cause misclassification in the rigorous *MV* setting, generated using our low-frequency (*LF*) strategy against all systems considered in §5.1. For configurations that were also evaluated against *AA*-ℓ_∞ under the **same** *MV* setting (i.e., using the same sample counts N as used for *LF* in §5.1), we include successful *AE*s generated with this method for direct comparison. Specifically, for ***ImageNet***, both the *LF* attack in §5.1 and the *AA* attack in §4.2 were evaluated under *MV* with $N = 8$; hence, we include *AA* samples for ***ImageNet*** directly from §4.2. For ***CIFAR-10***, the *AA* experiments in §4.2 use the more permissive $N = 10$ setting. Therefore, we instead draw samples from the corresponding $N = 128$ *AA* experiments reported in Appendix H to ensure a fair comparison under equal majority-vote conditions. Note that all samples are crafted using the parameters listed in §4.2 and §5.1. That is, $\tau_p = 0.05$ for *LF* and $\epsilon_\infty = 8/255$ for *AA* against ***CIFAR-10*** and $\epsilon_\infty = 4/255$ against ***ImageNet***.

For the configurations that were evaluated against both strategies, we provide two sets of samples: 1) Three triplets containing the original image, the *AE* generated using *AA*, and the *AE* crafted using *LF*. Importantly, all original samples in this set are inputs for which both methods can generate successful *AE*s, and we provide these to allow for a direct comparison between the two strategies' output quality on a sample-by-sample basis. Yet, as *AA* is inferior to our approach (*LF*), resulting in the systems retaining robustness on many inputs for which it fails to generate successful *AE*s under *MV*, it is essential to inspect *LF*'s outputs on such more challenging samples to demonstrate that it still preserves quality despite its ability to fool the target classifiers. Thus, we include a second set of 2) Three successful *AE*s generated with *LF* from inputs on which *AA* fails under *MV*. For the remaining configurations that were not evaluated against *AA* under the same *MV* sample counts from §5.1, we provide six successful *AE*s generated with *LF*.

In Appendix J.11, we present attack images generated by the non-norm-bounded *StAdv* [47] strategy under *MV* (with the above sample counts N). This method has demonstrated superior performance to norm-based techniques in the past against *DBP* [30] even in the absence of the correct exact gradients, indicating it could be a viable attack strategy with our gradient computation fixes, thereby making

our *LF* approach unnecessary. Yet, previous evaluations only considered *StAdv* against *DBP* for ***CIFAR-10*** [30]. While we find *StAdv* capable of defeating all systems (for both ***CIFAR-10*** and ***ImageNet***), it leads to severe quality degradation when used to attack *DBP*-protected classifiers for high-resolution inputs (i.e., ***ImageNet***), leaving the *AE*s of no utility. Thus, we deem it unsuitable, excluding it from the main body of the paper accordingly. Further details are in Appendix J.11.

All samples below are originally (without adversarial perturbations) correctly classified.

J.1 Attack Samples Generated Against *CIFAR-10*'s WideResNet-70-16 with *GDMP* Purification

Figure 3: Successful attacks generated by *LF* and *AA*-ℓ_∞. Left -original image. Middle - *AA*. Right - *LF*.

Figure 4: Successful *LF* attacks on inputs for which *AA*-ℓ_∞ fails. Left - original image. Right - *LF*.

J.2 Attack Samples Generated Against *CIFAR-10*'s WideResNet-28-10 with *GDMP* Purification

Note: While WideResNet-28-10 was not evaluated against AA under MV with $N = 128$ in Appendix H, we include AA samples here to complement our CIFAR-10 analyses—our primary benchmark for AA-based evaluation. Among the two main purification paradigms considered (GDMP and DiffPure), we randomly selected GDMP for this illustrative example.

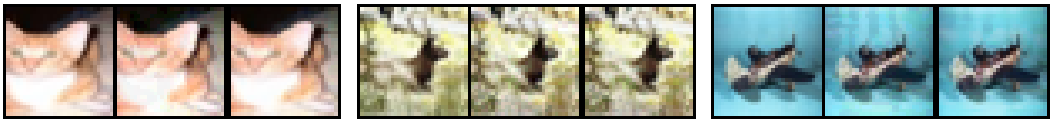


Figure 5: Successful attacks generated by *LF* and *AA*-ℓ_∞. Left -original image. Middle - *AA*. Right - *LF*.

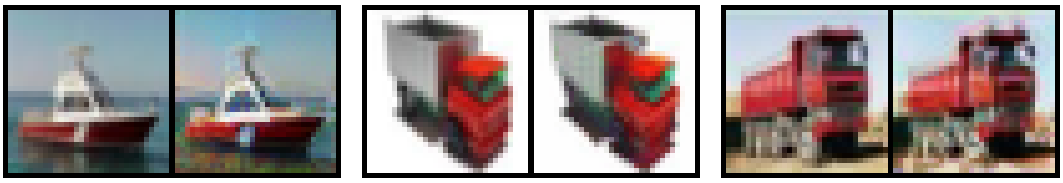


Figure 6: Successful *LF* attacks on inputs for which *AA*-ℓ_∞ fails. Left - original image. Right - *LF*.

J.3 Attack Samples Generated Against *CIFAR-10*'s WideResNet-70-16 with *DiffPure* Purification

Figure 7: Successful attacks generated by *LF* and *AA*-ℓ_∞. Left -original image. Middle - *AA*. Right - *LF*.

Figure 8: Successful *LF* attacks on inputs for which *AA*-ℓ_∞ fails. Left - original image. Right - *LF*.

J.4 Attack Samples Generated Against *CIFAR-10*'s WideResNet-28-10 with *DiffPure* Purification

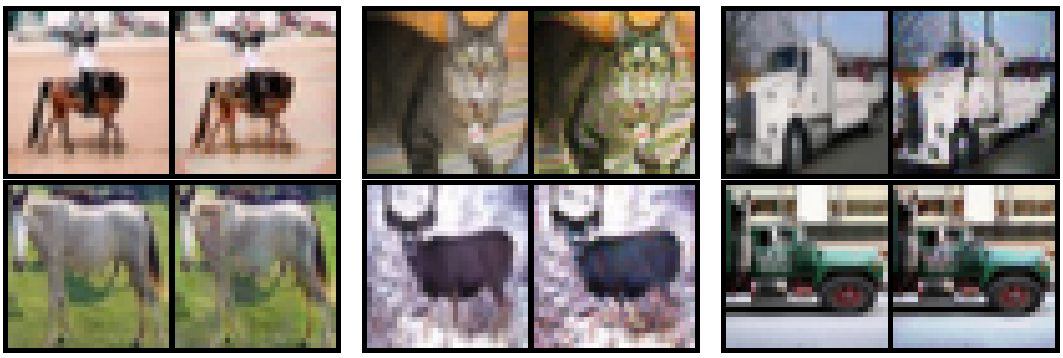


Figure 9: Successful attacks generated with *LF*. Left -original image. Right - *LF*.

J.5 Attack Samples Generated Against *ImageNet*'s DeiT-S with *GDMP* Purification

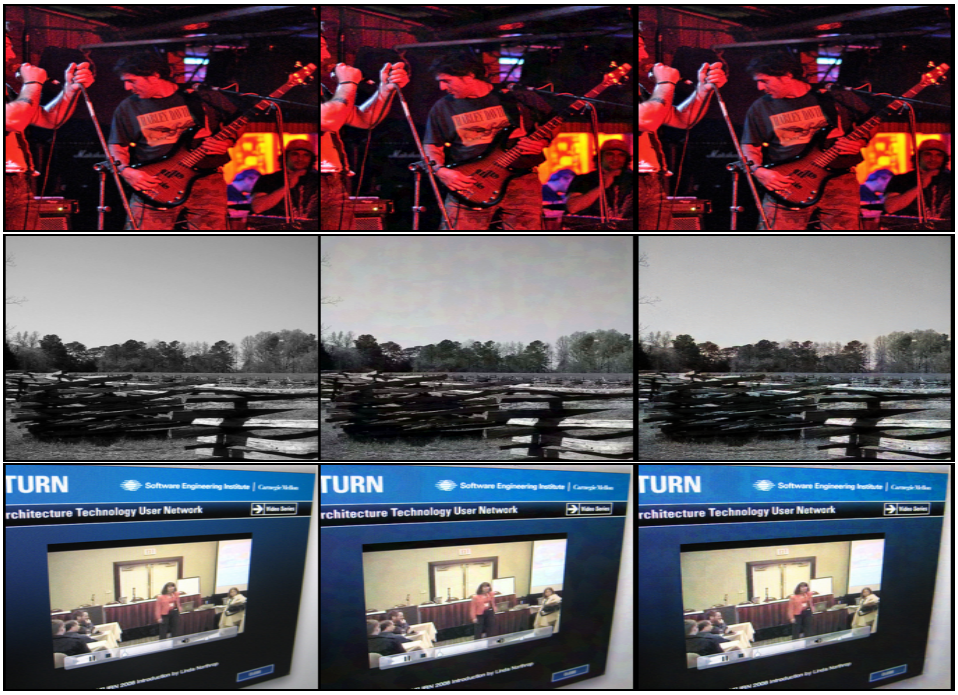


Figure 10: Successful attacks generated by *LF* and *AA*-ℓ_∞. Left -original image. Middle - *AA*. Right - *LF*.

Figure 11: Successful *LF* attacks on inputs for which *AA*-ℓ_∞ fails. Left - original image. Right - *LF*.

J.6 Attack Samples Generated Against *ImageNet*'s WideResNet-50-2 with *GDMP* Purification

Figure 12: Successful attacks generated with *LF*. Left -original image. Right - *LF*.

J.7 Attack Samples Generated Against *ImageNet*'s ResNet-50 with *GDMP* Purification

Figure 13: Successful attacks generated with *LF*. Left -original image. Right - *LF*.

J.8 Attack Samples Generated Against *ImageNet*'s DeiT-S with *DiffPure* Purification

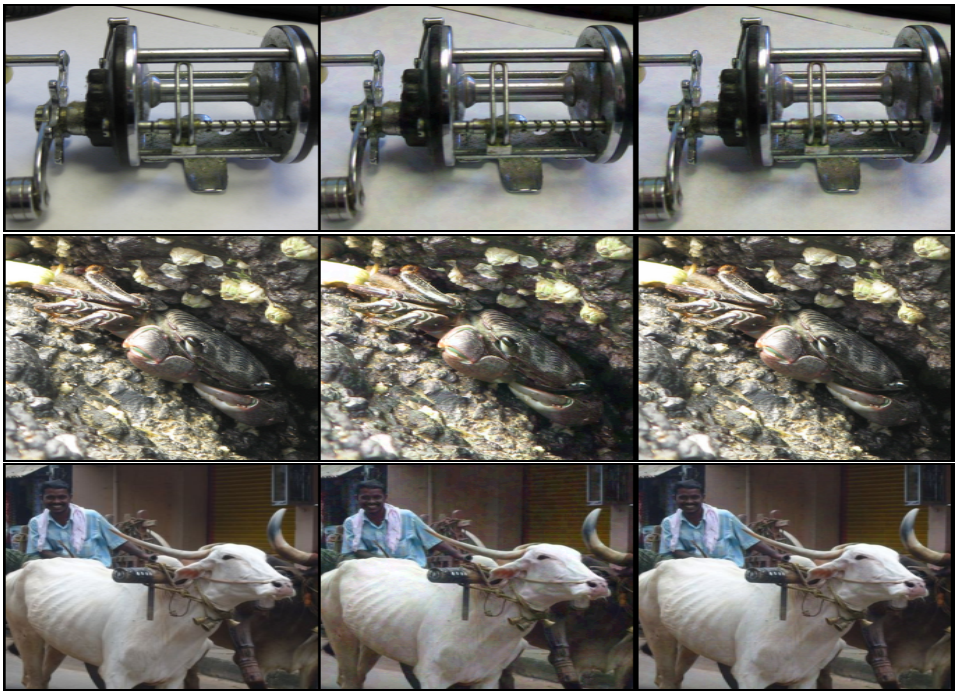


Figure 14: Successful attacks generated by *LF* and *AA*-ℓ_∞. Left -original image. Middle - *AA*. Right - *LF*.

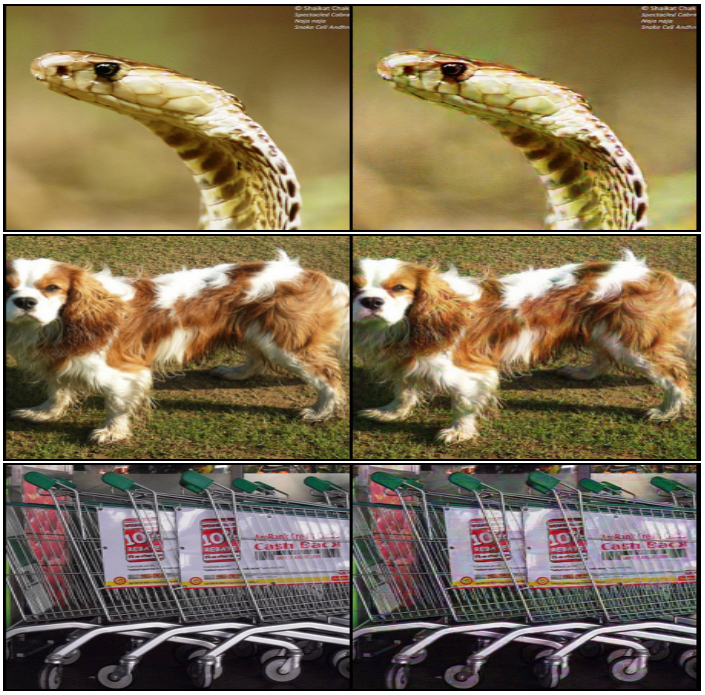


Figure 15: Successful *LF* attacks on inputs for which *AA*-ℓ_∞ fails. Left - original image. Right - *LF*.

J.9 Attack Samples Generated Against *ImageNet*'s WideResNet-50-2 with *DiffPure* Purification

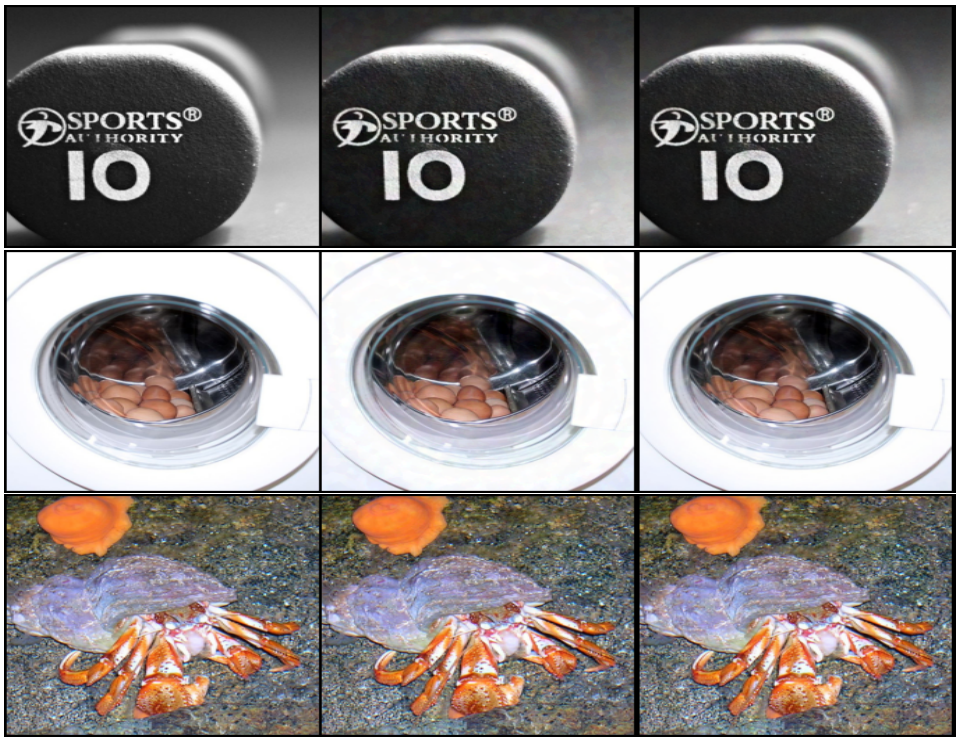


Figure 16: Successful attacks generated by *LF* and *AA*-ℓ_∞. Left -original image. Middle - *AA*. Right - *LF*.

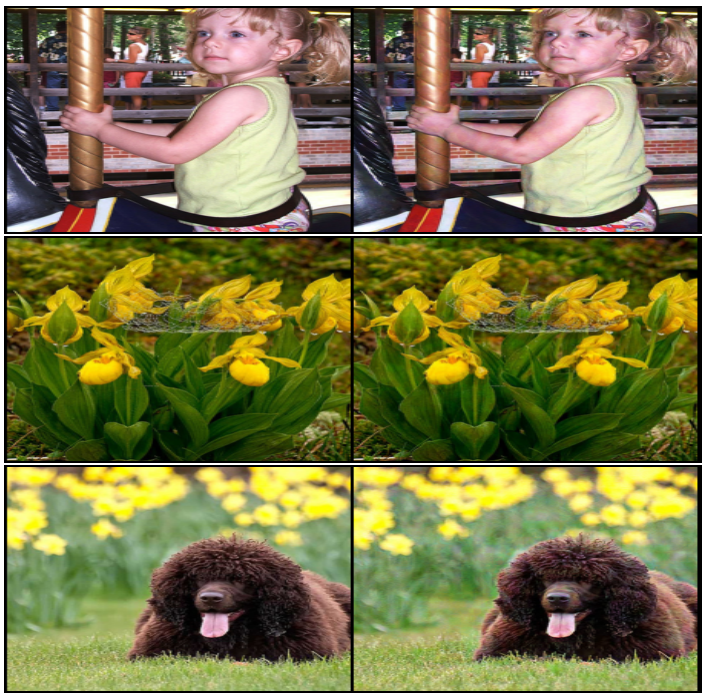


Figure 17: Successful *LF* attacks on inputs for which *AA*-ℓ_∞ fails. Left - original image. Right - *LF*.

J.10 Attack Samples Generated Against *ImageNet*'s ResNet-50 with *DiffPure* Purification

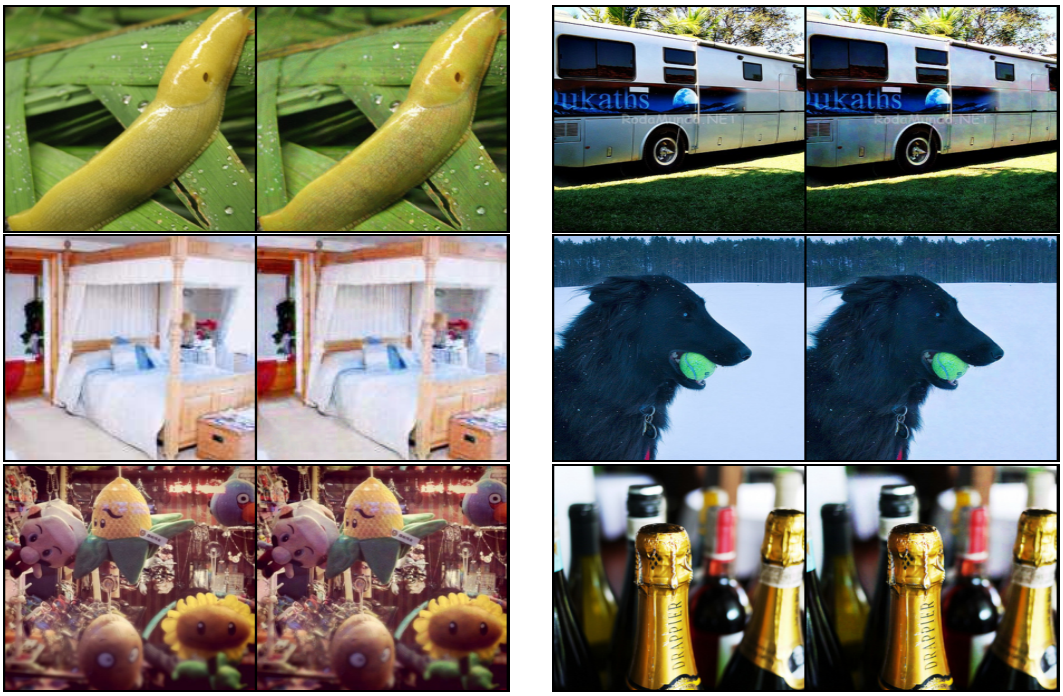


Figure 18: Successful attacks generated with *LF*. Left -original image. Right - *LF*.

J.11 Quality Comparison with *StAdv*

We found *StAdv* capable of generating outputs that defeat *DBP* even under *MV*. However, it is not suitable for targeting *DBP*-defended classifiers that operate on high-resolution images. The reason is that *StAdv* performs spatial transformations that relocate the different pixels. Thus, its changes quickly become visible when applied excessively. Due to the considerable stochasticity of *DBP* (see §4.2), the required displacements (especially in the *MV* setting) are significant, which in turn can severely impact the quality. For low-resolution inputs (e.g., ***CIFAR-10***), *StAdv* can still be effective, with the quality degradation remaining unnoticeable due to the size of the images that renders them blurry by default, masking *StAdv*'s effects. For high-resolution inputs, the degradation is substantial, leaving the outputs useless as stealthiness is a key requirement from practical *AE*s [21]. *StAdv*'s successfully misclassified samples (under *MV*) below prove these claims. We use **Full-DiffGrad** for backpropagation and run *StAdv* with its default parameters [47]. When the parameters are changed to better retain quality, *StAdv* ceases to converge for ***ImageNet***, making it of no use. All provided samples are originally correctly classified.

Figure 19: *StAdv* attacks against ***CIFAR-10***'s WideResNet-70-16 with *GDMP* purification. Left - original image. Right - *StAdv*.

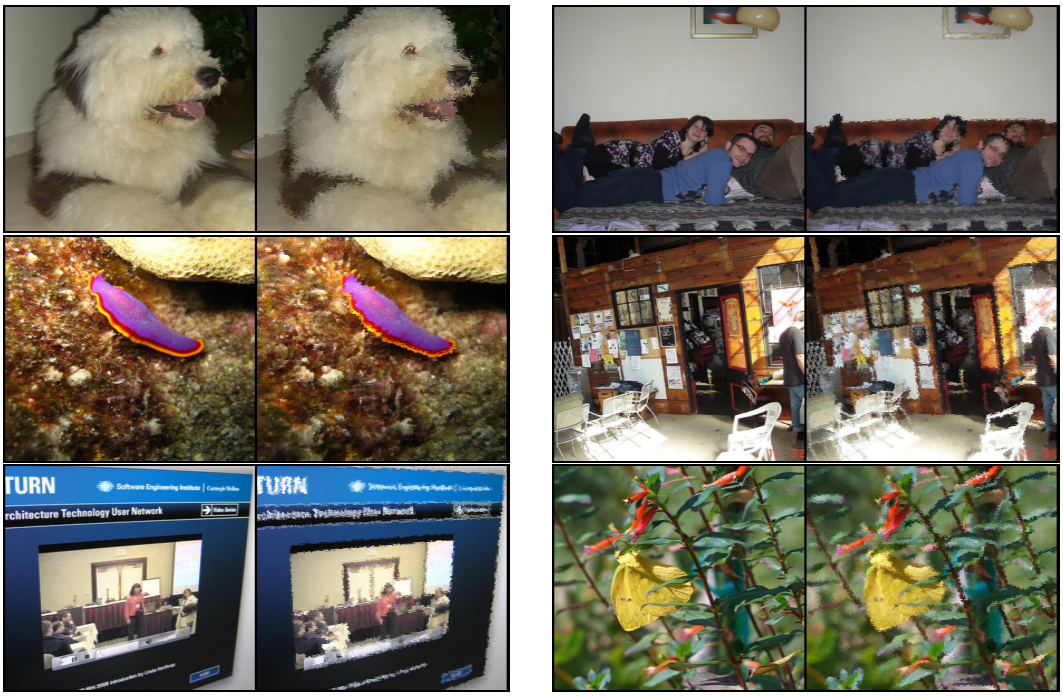


Figure 20: *StAdv* attacks against ***ImageNet***'s DeiT-S with *DiffPure* purification. Left - original image. Right - *StAdv*.