Appendix

A. Inference Loss Function

	Food101	CIFAR10	Aircraft	Oxford Pets	Flowers102	STL10	ImageNet	ObjectNet
Squared ℓ_2	**77.9**	76.3	**24.3**	85.7	56.8	94.2	58.4	**38.3**
ℓ_1	74.3	**87.1**	18.3	**86.2**	**59.4**	**95.3**	58.0	32.2
Huber	**77.9**	76.9	23.7	85.5	57.5	94.2	**58.9**	38.1

Table 4. **Diffusion Classifier zero-shot performance with different loss functions** $\mathcal{L}(\epsilon - \epsilon_\theta(\mathbf{x}_t, \mathbf{c}))$.

	ImageNet	ImageNetV2	ImageNet-A
Squared ℓ_2	**77.3**	**64.5**	**19.6**
ℓ_1	74.4	60.8	9.4

Table 5. **Diffusion Classifier supervised performance with different loss functions** $\mathcal{L}(\epsilon - \epsilon_\theta(\mathbf{x}_t, \mathbf{c}))$.

While the theory in Section 3 justifies using $\|\epsilon - \epsilon_\theta(\mathbf{x}_t, \mathbf{c})\|_2^2$ within the Diffusion Classifier inference objective, we surprisingly find that other loss functions can work better in some cases. Table 4 shows that using the ℓ_1 loss instead of the squared ℓ_2 loss to evaluate the magnitude of $\epsilon - \epsilon_\theta(\mathbf{x}_t, \mathbf{c})$ does better on roughly half of the datasets that we use to evaluate the Stable Diffusion-based zero-shot classifier. This is puzzling, since the ℓ_1 loss is neither theoretically justified nor appears in the Stable Diffusion training objective. We hope followup work can explain the empirical success of the ℓ_1 loss.

Combining these two losses does not get the “best of both worlds.” The Huber loss, which is the squared ℓ_2 loss for values less than 1 and is the ℓ_1 loss for values greater than 1, roughly achieves the same performance as the theoretically-justified squared ℓ_2 loss. Table 5 shows that ℓ_1 does not help with supervised classification (Section 6.4) using DiT-XL/2 at 256×256.

B. Inference Costs and Hybrid Classification Approach

	Food101	CIFAR10	Aircraft	Oxford Pets	Flowers102	STL10	ImageNet
Diffusion Classifier	77.9	76.3	24.3	85.7	56.8	94.2	58.4
Time/img (s)	51	30	51	18	51	30	1000
Diffusion Classifier w/ Prune	78.9	76.3	24.6	85.4	59.8	94.2	60.5
Time/img (s)	35	30	35	18	35	30	150
Est. Time/img (s) at 128^2 res	2	2	2	1	2	2	9
CLIP ResNet-50	81.1	75.6	19.3	85.4	65.9	94.3	58.2

Table 6. **Zero-shot accuracy and inference time with Stable Diffusion** 512×512**.** “Pruning” away unlikely classes with a weak discriminative classifier (e.g., CLIP ResNet-50) increases accuracy and reduces inference time. Additionally, reducing resolution to 128×128 would reduce inference time by roughly $16\times$. However, its impact on accuracy is difficult to estimate without retraining the Stable Diffusion model to expect lower resolutions. All times are estimated using a RTX 3090 GPU.

Table 1 shows the inference time of Diffusion Classifier when using the efficient Diffusion Classifier algorithm (Algorithm 2). Classifying a single image takes anywhere between 18 seconds (Pets) to 1000 seconds (ImageNet). The issue with ImageNet is that Diffusion Classifier inference time still approximately scales linearly with the number of classes, even when using the adaptive strategy. One way to address this problem is to use a weak discriminative model to quickly “prune” away classes that are almost certainly incorrect. Table 1 shows that using CLIP ResNet-50 to restrict the search to the top 20 classes greatly reduces inference time, while even improving performance. This works even when the top-1 accuracy of the ResNet-50 is low, like on Aircraft.

C. Techniques that did not help

Diffusion Classifier requires many samples to accurately estimate the ELBO. In addition to using the techniques in Section 3 and 4, we tried several other options for variance reduction. None of the following methods worked, however. We list negative results here for completeness, so others do not have to retry them.

Classifier-free guidance Classifier-free guidance [36] is a technique that improves the match between a prompt and generated image, at the cost of mode coverage. This is done by training a conditional $\epsilon_\theta(\mathbf{x}_t, \mathbf{c})$ and unconditional $\epsilon_\theta(\mathbf{x}_t)$ denoising network and combining their predictions at sampling time:

$$\tilde{\epsilon}(\mathbf{x}_t, \mathbf{c}) = (1 + w)\epsilon_\theta(\mathbf{x}_t, \mathbf{c}) - w\epsilon_\theta(\mathbf{x}_t) \tag{10}$$

where w is a guidance weight that is typically in the range $[0, 10]$. Most diffusion models are trained to enable this trick by occasionally replacing the conditioning $\mathbf{c}$ with an empty token. Intuitively, classifier-free guidance "sharpens" $\log p_\theta(x \mid \mathbf{c})$ by encouraging the model to move away from regions that unconditionally have high probability. We test Diffusion Classifier to see if using the $\tilde{\epsilon}$ from classifier-free guidance can improve confidence and classification accuracy. Our new ϵ-prediction metric is now $\|\epsilon - (1 + w)\epsilon_\theta(\mathbf{x}_t, \mathbf{c}) - w\epsilon_\theta(\mathbf{x}_t)\|^2$. However, Figure 8 shows that $w = 0$ (i.e., no classifier-free guidance) performs best. We hypothesize that this is because Diffusion Classifier fails on uncertain examples, which classifier-free guidance affects unpredictably.

Error map cropping The ELBO $\mathbb{E}_{t,\epsilon}[\|\epsilon - \epsilon_\theta(\mathbf{x}_t, \mathbf{c})\|^2]$ depends on accurately estimating the added noise at every location of the $64 \times 64 \times 4$ image latent. We try to reduce the impact of edge pixels (which are less likely to contain the subject) by computing $\mathbf{x}_t$ as normal, but only measuring the ELBO on a center crop of ϵ and $\epsilon_\theta(\mathbf{x}_t, \mathbf{c})$. We compute:

$$\|\epsilon_{[i:-i,i:-i]} - \epsilon_\theta(\mathbf{x}_t, \mathbf{c})_{[i:-i,i:-i]}\|^2 \tag{11}$$

where i is the number of latent "pixels" to remove from each edge. However, Figure 9 shows that any amount of cropping reduces accuracy.

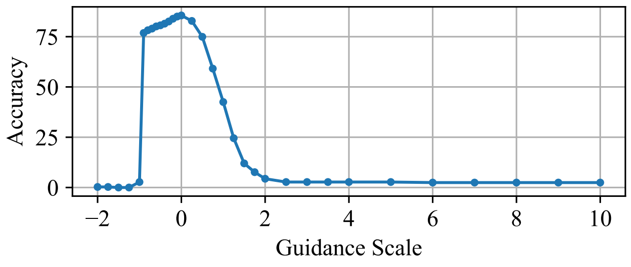

Figure 8. Accuracy plot of classifier-free guidance on Pets.

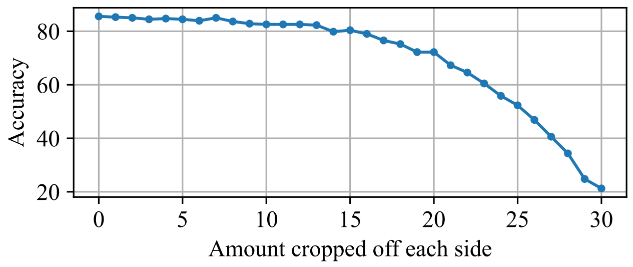

Figure 9. Cropping ϵ and $\epsilon_\theta(\mathbf{x}_t, \mathbf{c})$ reduces accuracy on Pets.

Importance sampling Importance sampling is a common method for reducing the variance of a Monte Carlo estimate. Instead of sampling $\epsilon \sim \mathcal{N}(0, I)$, we sample ϵ from a narrower distribution. We first tried fixing $\epsilon = 0$, which is the mode of $\mathcal{N}(0, I)$, and only varying the timestep t. We also tried the truncation trick [7] which samples $\epsilon \sim \mathcal{N}(0, I)$ but continually resamples elements that fall outside the interval $[a, b]$. Finally, we tried sampling $\epsilon \sim \mathcal{N}(0, I)$ and rescaling them to the expected norm ($\epsilon \to \frac{\epsilon}{\|\epsilon\|_2}\mathbb{E}_{\epsilon'}[\|\epsilon'\|_2]$)) so that there are no outliers. Table 7 shows that none of these importance sampling strategies improve accuracy. This is likely because the noise ϵ sampled with these strategies are completely out-of-distribution for the noise prediction model. For computational reasons, we performed this experiment on a 10% subset of Pets.

D. Additional Implementation Details

D.1. Efficient Diffusion Classifier Algorithm

Though Diffusion Classifier works straightforwardly with the procedure described in Algorithm 1, we are interested in speeding up inference as described in Section 4.2. Algorithm 2 shows the efficient Diffusion Classifier procedure that adaptively chooses which classes to continue evaluating.

Sampling distribution for ϵ	Pets accuracy
$\epsilon = 0$	41.3
TruncatedNormal, $[-1, 1]$	49.9
TruncatedNormal, $[-2.5, 2.5]$	81.5
Expected norm	86.9
$\epsilon \sim \mathcal{N}(0, I)$	87.5

Table 7. Every importance sampling strategy underperforms sampling the noise ϵ from a standard normal distribution.

Algorithm 2 `Efficient Diffusion Classifier`

```
1: Input: test image $\mathbf{x}$, conditioning inputs $\{\mathbf{c}_i\}_{i=1}^n$ (e.g., text embeddings or class indices), number of stages $N_{\text{stages}}$, list KeepList of number of $\mathbf{c}_i$ to keep after each stage, list TrialList of number of trials done by each stage
2: Initialize Errors$[\mathbf{c}_i]$ = list() for each $\mathbf{c}_i$
3: $\mathcal{C} = \{\mathbf{c}_i\}_{i=1}^n$
4: PrevTrials = 0
5: for stage i $= 1, \ldots, N_{\text{stages}}$ do
6:     for trial $j = 1, \ldots,$ TrialList$[i]$ − PrevTrials do
7:         Sample $t \sim [1, 1000]$
8:         Sample $\epsilon \sim \mathcal{N}(0, I)$
9:         $\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\mathbf{x} + \sqrt{1 - \bar{\alpha}_t}\epsilon$
10:        for conditioning $\mathbf{c}_k \in \mathcal{C}$ do
11:            Errors$[\mathbf{c}_k]$.append($\|\epsilon - \epsilon_\theta(\mathbf{x}_t, \mathbf{c}_k)\|^2$)
12:        end for
13:    end for
14:    // Keep best KeepList$[i]$ $\mathbf{c}_k$ with the lowest errors
15:    $\mathcal{C} \leftarrow \underset{\substack{\mathcal{S} \subset C; \\ |\mathcal{S}| = \texttt{KeepList}[i]}}{\arg\min} \sum_{c_k \in \mathcal{S}}$ mean(Errors$[\mathbf{c}_k]$)
16:    PrevTrials = TrialList[i]
17: end for
18: return $\underset{\mathbf{c}_i \in \mathcal{C}}{\arg\min}$ mean(Errors$[\mathbf{c}_i]$)
```

D.2. Zero-shot classification using Diffusion Classifier

Training Data For our zero-shot Diffusion Classifier, we utilize Stable Diffusion 2.1 [64]. This model was trained on a subset of the LAION-5B dataset, filtered so that the training data is aesthetic and appropriately safe-for-work. LAION classifiers were used to remove samples that are too small (less than 256×256), potentially pornographic (punsafe ≥ 0.1), or unaesthetic (aesthetic score < 4.5). These thresholds are relatively conservative, since false negatives (NSFW or undesirable images left in the training set) are worse than removing extra images from a large starting dataset. As discussed in Section 6.1, these filtering criteria bias the distribution of Stable Diffusion training data and likely negatively affect Diffusion Classifier's performance on datasets whose images do not satisfy these criteria. The checkpoint we use was trained for 550k steps at resolution 256×256 on this subset, followed by an additional 850k steps at resolution 512×512 on images that are at least that large. This checkpoint can be downloaded online through the diffusers repository at `stabilityai/stable-diffusion-2-1-base`.

Inference Details We use FP16 and Flash Attention [15] to improve inference speed. This enables efficient inference with a batch size of 32, which works across a variety of GPUs, from RTX 2080Ti to A6000. We found that adding these two tricks did not affect test accuracy compared to using FP32 without Flash Attention. Given a test image, we resize the shortest edge to 512 pixels using bicubic interpolation, take a 512×512 center crop, and normalize the pixel values to $[-1, 1]$. We then use the Stable Diffusion autoencoder to encode the $512 \times 512 \times 3$ RGB image into a $64 \times 64 \times 4$ latent. We finally classify the test image by applying the method described in Sections 3 and 4 to estimate ϵ-prediction error in this latent space.

Dataset	Prompts kept per stage	Evaluations per stage	Avg. evaluations per class	Total evaluations
Food101	20 10 5 1	20 50 100 500	50.7	5120
CIFAR10	5 1	100 500	300	3000
FGVC Aircraft	20 10 5 1	20 50 100 500	51	5100
Pets	5 1	25 250	51	1890
Flowers102	20 10 5 1	20 50 100 500	50.4	5140
STL10	5 1	100 500	300	3000
ImageNet	500 50 10 1	50 100 500 1000	100	100000
ObjectNet	25 10 5 1	50 100 500 1000	118.6	13400

Table 8. Evaluation strategy for each zero-shot dataset.

Resolution	ImageNet accuracy (%)
256×256	77.3
512×512	80.1

Table 9. Diffusion Classifier ImageNet accuracy improves when using higher resolution DiT models.

Sampling Strategy Table 8 shows the evaluation strategy used for each zero-shot dataset. We hand-picked the strategies based on the number of classes in each dataset. Further gains in accuracy may be possible with more evaluations.

D.3. Compositional reasoning using Diffusion Classifier

For our experiments on the Winoground benchmark [74], most details are the same as in Appendix D.2. We use Stable Diffusion 2.1, and we evaluate each image-caption pair with 250 timesteps. We omit the adaptive inference strategy since there are only 4 image-caption pairs to evaluate for each Winoground example.

D.4. ImageNet classification using Diffusion Classifier

For this task, we use the recent Diffusion Transformer (DiT) [57] as the backbone of our Diffusion Classifier. DiT was trained on ImageNet-1k, which contains about 1.28 million images from 1,000 classes. While it was originally trained to produce high-quality samples with strong FID scores, we repurpose the model and compare it against discriminative models with the same ImageNet-1k training data. We use the DiT-XL/2 model sizes. Inference with DiT-XL/2 at resolution 512×512 is about 4 times more expensive than 256×256, but it performs better (see Table 9). Notably, DiT achieves strong performance while using much weaker data augmentations than what discriminative models are usually trained with. During training time for the 256×256 checkpoint, the smaller edge of the input image is resized to 256 pixels. Then, a 256×256 center crop is taken, followed by a random horizontal flip, followed by embedding with the Stable Diffusion autoencoder. At test time, we follow the same preprocessing pipeline, but omit the random horizontal flip. Classification performance could improve if stronger augmentations, like RandomResizedCrop or color jitter, are used during the diffusion model training process.

D.5. Baselines for Zero-Shot Classification

Synthetic-SD: We provide the implementation details of the "Synthetic-SD" baseline (row 1 of Table 1) for the task of zero-shot image classification. Our Diffusion Classifier approach builds on the intuition that a model capable of generating examples of desired classes should be able to directly discriminate between them. In contrast, this baseline takes the simple approach of using our generative model, Stable Diffusion, as intended to generate *synthetic training data* for a discriminative model. For a given dataset, we use pre-trained Stable Diffusion 2.1 with default settings to generate 10,000 synthetic 512×512 pixel images per class as follows: we use the English class name and randomly sample a template from those provided by the CLIP [59] authors to form the prompt for each generation. We then train a supervised ResNet-50 classifier using the synthetic data and the labels corresponding to the class name that was used to generate each image. We use batch size $= 256$, weight decay $= 1e-4$, learning rate $= 0.1$ with a cosine schedule, the AdamW optimizer, and use random resized crop & horizontal flip transforms. We create a validation set using the synthetic data by randomly selecting 10% of the images for each class; we use this for early stopping to prevent over-fitting. Finally, we report the accuracy on the target dataset's proper test set.

Arch	Conv1	Conv2	Conv3 x2	Conv4 x2	Conv5 x2
ResNet-18	7x7x64	3x3 max-pool	3x3x128	3x3x256	3x3x512
ResNet-18 (Real-Labeled-SD)	3x3x1280	-	3x3x1280	3x3x2560	3x3x2560

Table 10. Comparison of Real-Labeled-SD's ResNet-18 classifier architecture with the original ResNet-18

Real-Labeled-SD: We provide the implementation details of the "Real-Labeled-SD" baseline (row 2 of Table 1) for the task of image classification. This baseline is inspired by Label-DDPM [3], a recent work on diffusion-based semantic segmentation. Unlike Label-DDPM, which leverages a category-specific diffusion model, we directly build on top of the open-sourced Stable Diffusion model (trained on the LAION dataset). We then approach the task of classification as follows: given the pre-trained Stable Diffusion model, we extract the intermediate U-Net features corresponding to the input image. These features are then passed through a ResNet-based classifier to predict the corresponding class name. To extract the intermediate U-Net features, we add a noise equivalent to the $100th$ timestep noise to the input image and evaluate the corresponding noisy latent using the forward diffusion process. We then pass the noisy latent through the U-Net model, conditioned on timestep $t = 100$ and text conditioning (y) as an empty string, and extract out the features from the mid-layer of the U-Net at a resolution of [8 × 8 × 1024]. Next, we train a supervised classifier on top of these features. *Thus, this baseline is not zero-shot.* The architecture of our classifier is similar to ResNet-18, with small modifications to make it compatible with an input size of $[8 \times 8 \times 1024]$. Table 10 defines these modifications. We set batch size $= 16$, learning rate $= 1e - 4$, and use AdamW optimizer. During training, we do augmentations similar to the original ResNet (Random Crop and Flip). We do early stopping using the validation set to prevent over-fitting. We use the official train-test splits for each dataset, except ImageNet and ObjectNet. For these two datasets, we perform class sub-sampling and use the same train-test split as our model. We do this to achieve fair comparisons with the other baselines.