A Model Details.

Imagen details: Imagen is a text-conditioned diffusion model that comprises of a frozen T5 (Raffel et al., 2020) language encoder that encodes an input prompt into a sequence of embeddings, a 64×64 image diffusion model, and two cascaded super-resolution diffusion models that generate 256×256 and 1024×1024 images. Unlike Stable Diffusion, it operates directly on pixels instead of in a latent space. For our experiments, we use the 64×64 model which has 2B parameters and is trained using a batch size of 2048 for 2.5M training steps on a combination of internal datasets, with around 460M image-text pairs, and the publicly available Laion dataset (Schuhmann et al., 2021), with 400M image-text pairs.

Stable Diffusion details. We use Stable diffusion v1.4 which is a latent text-to-image diffusion model. It uses the pre-trained text encoder from CLIP to encode text and a pre-trained variational autoencoder to map images to a latent space. The model has 890M parameters and takes 512x512-resolution images as input. It was trained on various subsets of Laion-5B, including a portion filtered to only contain aesthetic images, for 1.2M steps using batch size of 2048.

CLIP details: CLIP encodes image features using a ViT-like transformer (Dosovitskiy et al., 2021) and uses a causal language model to get the text features. After encoding the image and text features to a latent space with identical dimensions, it evaluates a similarity score between these features. CLIP is pre-trained using contrastive learning. Here, we compare to the largest CLIP model (with a ViT-L/14@224px as the image encoder). The model is smaller than Imagen (400M parameters), but is trained for longer (12.8B images processed vs 5.B). While Imagen was trained primarily as a generative model, CLIP was primarily engineered to be transferred effectively to downstream tasks.

B Weighting Functions Details.

Learned Weighting Function: While for most experiments we use a heuristic weighting function for $\boldsymbol{w}_t$, we also explored learning an effective weighting function (although this is not truly zero-shot). To do this, we aggregate scores for each image $\boldsymbol{x}$ and class y_k into 20 buckets, with each bucket covering a small slice of timestep values:

$$\boldsymbol{b}_i(\boldsymbol{x}, \mathrm{y}_k) = \mathbb{E}_{\epsilon, t \sim \mathcal{U}[0.05i, 0.05(i+1)]} \|\boldsymbol{x} - \tilde{\boldsymbol{x}}_{\boldsymbol{\theta}}\big(\boldsymbol{x}_t, \boldsymbol{\phi}(\mathrm{y}_k), t\big)\|_2^2$$

where we estimate the expectation with Monte Carlo sampling (typically around 100 samples). We then learn a 20-feature linear model with parameters $[\boldsymbol{v}_0, ..., \boldsymbol{v}_{19}]$ over these buckets:

$$p_{\boldsymbol{v}}(y = \mathrm{y}_k | \boldsymbol{x}) = \frac{\exp(\sum_{i=0}^{19} -\boldsymbol{v}_i \boldsymbol{b}_i(\boldsymbol{x}, \mathrm{y}_k))}{\sum_{\mathrm{y}_j \in [\mathrm{y}_K]} \exp(\sum_{i=0}^{19} -\boldsymbol{v}_i \boldsymbol{b}_i(\boldsymbol{x}, \mathrm{y}_j))}$$

trained with standard maximum likelihood over the data. At test-time we use the weighting

$$\boldsymbol{w}_t = \boldsymbol{v}_{\lfloor t/0.05 \rfloor}$$

We generally found that (1) learned weighting functions are pretty similar across datasets, and (2) the weighting functions are transferable: the $\boldsymbol{v}$s learned on one dataset get good accuracy when evaluated on other ones. On average, learned weights produced around 1% higher accuracy on zero-shot classification tasks, but we omitted the results from the main paper because using learned weights is not truly zero-shot.

Comparison of Weighting Functions. We compare the learned weighting functions with several heuristic functions on the Caltech101 dataset. We chose Caltech101 because it is high-resolution (SD performs poorly on low-resolution datasets), contains a diversity of image classes, was not used when we developed the heuristic weighting function, and only has 100 classes, so it is much faster to evaluate models on than ImageNet. We compare the following functions:

- **VDM**: $\boldsymbol{w}_t = \mathrm{SNR}'(t)$, the derivative of the signal to noise ratio with respect to t. This weighting scheme from Variational Diffusion Models (Kingma et al., 2021) directly trains the model on a lower bound of the likelihood.
- **Simple**: $\boldsymbol{w}_t = \mathrm{SNR}(t)$. This "simple" loss from Ho et al. (2020) results in a model that produces better images according to human judgements and FID scores, even though it results in worse likelihoods.

Weighting	Imagen	Stable Diffusion
VDM	62.0	71.9
Simple	56.1	71.4
Heuristic	68.9	73.0
Learned	70.2	73.1

Table 4: Percent accuracy for models on Caltech101 with different weighting schemes

- **Heuristic**: $\boldsymbol{w}_t = \exp(-6t)$. Our hand-engineered weighting function; we found this by searching for a simple weighting function that works well on CIFAR-100, although we found empirically it generalizes very well to other datasets.
- **Learned**: Learning an effective weighting function on a held-out set of examples as described above.

Results are shown in Table 4. The heuristic weighting function outperforms Simple and VDM for both models. Interestingly, SD appears to be more robust to the choice of weighting function than Imagen. Mechanistically, the reason is that "Simple" and "VDM" weighting put more weight on earlier timesteps than "Heuristic" and Imagen tends to be an inaccurate classifier at very small noise levels. We intuitively believe this is a consequence of pixel vs latent diffusion. The learned weighting only does slightly better than heuristic weighting despite not being truly zero-shot. We found similar results to hold on other datasets.

C Details on Attribute Binding Tasks and Prompts

We use the relational dataset from Lewis et al. (2022) for the attribute binding experiments. Each image consists of two objects of different shapes and colors; for tasks involving size we filter out examples where both objects are the same size. Each image contains two objects with different attributes $\mathsf{shape} \in \{\text{cube}, \text{sphere}, \text{cylinder}\}$, $\mathsf{color} \in \{\text{blue}, \text{cyan}, \text{blue}, \text{brown}, \text{gray}, \text{green}, \text{purple}, \text{red}, \text{yellow}\}$, $\mathsf{size} \in \{\text{small}, \text{large}\}$, and $\mathsf{position} \in \{\text{left}, \text{right}\}$.

Given a task (e.g. Shape|Size), we construct a task-specific description for an object as follows:

*"On the {*position*} is a "* if Position tasks else *"A "*+
"size " if Size task else ""+
"color " if Color task else ""+
"shape." if Shape task else *"object."*

For recognition and binding tasks, we randomly select one of the two objects in the image to be the positive example and then use its description as the positive prompt. For pair tasks, we join the descriptions for both objects together with "and" (removing the period from the first description and lowercasing the second one) for the positive prompt.

To construct a negative example for recognition tasks, we replace the positive attribute with a random attribute not in the image. For binding tasks, we replace one of positive description's attributes with the other object's attribute (e.g., for Shape|Color, we replace shape).

For pair tasks, there is a choice in how the two objects are ordered (e.g. "On the left is a cube and on the right is a sphere" vs "On the right is a sphere and on the left is a cube". We follow the preference of stating the leftmost position/shape/color/size first in that order. For example, this means we will always start with "On the left..." rather than "On the right...". Similarly, the negative example for Color,Size in Figure 4 is "A large yellow object and small gray object" rather than "A small gray object and a large yellow object" because we prefer to first put the leftmost color over the leftmost size.

We experimented with a variety of other prompts, but found none to work substantially better than these simple ones.

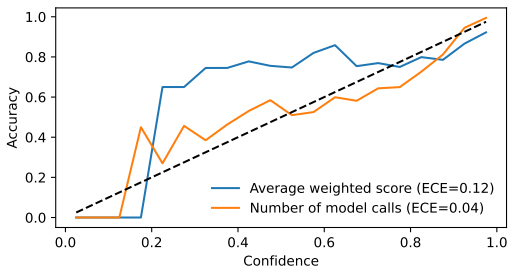

Figure 5: Model reliability diagram comparing confidence measures of Imagen on CIFAR-100. The number of model calls used in Algorithm 1 produces better-calibrated confidences than using the actual scores for different classes.

D Calibration

It is desirable for classifiers, especially when used in the zero-shot setting with possibly out-of-domain examples, to be well calibrated. In other words, if a classifier predicts a label $\tilde{y}_i$ with probability p, the true label should be $\tilde{y}_i$ roughly $100 \cdot p\%$ of the time. However, the diffusion model classifier does not directly produce probabilities for classes. While $p(y_i = \mathrm{y}_k | \boldsymbol{x}_i)$ should roughly be proportional to the expectation in Equation (2) when exponentiated (i.e. we can apply a softmax to the average weighted scores to get probabilities), in practice our estimates of the expectations are very noisy and do not provide well-calibrated scores.

We propose a simple alternative that takes advantage of early pruning: we use the total number of diffusion model calls used for the image as a calibration measure. The intuition is that a harder example will require more scores to determine the $\arg\min$ class with good statistical significance.

More details on the two calibration methods are below:

Temperature-scaled raw scores. We use $s_{\mathrm{y}_k}(\boldsymbol{x})$ to denote the weighted average squared error for class y_k on image $\boldsymbol{x}$, i.e., the Monte-Carlo estimate for the re-weighted VLB in equation 2. We turn these scores into an estimated probability by applying a softmax with temperature:

$$p_\theta(y = \mathrm{y}_k | \boldsymbol{x}) = \frac{\exp(-s_{\mathrm{y}_k}(\boldsymbol{x})/\tau)}{\sum_{\mathrm{y}_j \in [\mathrm{y}_K]} \exp\left(-s_{\mathrm{y}_j}(\boldsymbol{x})/\tau\right)}$$

Note that this approach requires good score estimates for each class, so it is not compatible with the class pruning method presented in Section 3.1.

Platt-scaled number of scores. Our other confidence method relies on the total number of scores needed to eliminate all other classes as candidates. Let $\tilde{y}(\boldsymbol{x})$ denote the predicted class for example $\boldsymbol{x}$ and $n(\boldsymbol{x})$ be the total number of calls to $\tilde{\boldsymbol{x}}_\theta$ used to obtain the prediction when running Algorithm 1. Then we estimate

$$p_\theta(y = \tilde{y}(\boldsymbol{x}) | \boldsymbol{x}) = \mathrm{sigmoid}(-n(\boldsymbol{x})/\tau + \beta)$$

We learn τ (and β for Platt scaling) on a small held-out set of examples.

We show reliability diagrams (DeGroot & Fienberg, 1983) and report Expected Calibration Error (Guo et al., 2017) (ECE) for the methods in Figure 5. Using a small held-out set of examples, we apply temperature scaling (Guo et al., 2017) for the score-based confidences and Platt scaling (Platt et al., 1999) for the number-of-scores confidences, (see Appendix D for details). Number of scores is fairly well-calibrated, showing it is possible to obtain reasonable confidences from diffusion model classifiers despite them not providing a probability distribution over classes.

E Imagen's Super-resolution Models

Imagen is a cascaded diffusion model (Ho et al., 2022) consisting of a 64×64 low-resolution model and two super-resolution models, one that upsample the image from 64×64 to 256×256 and one

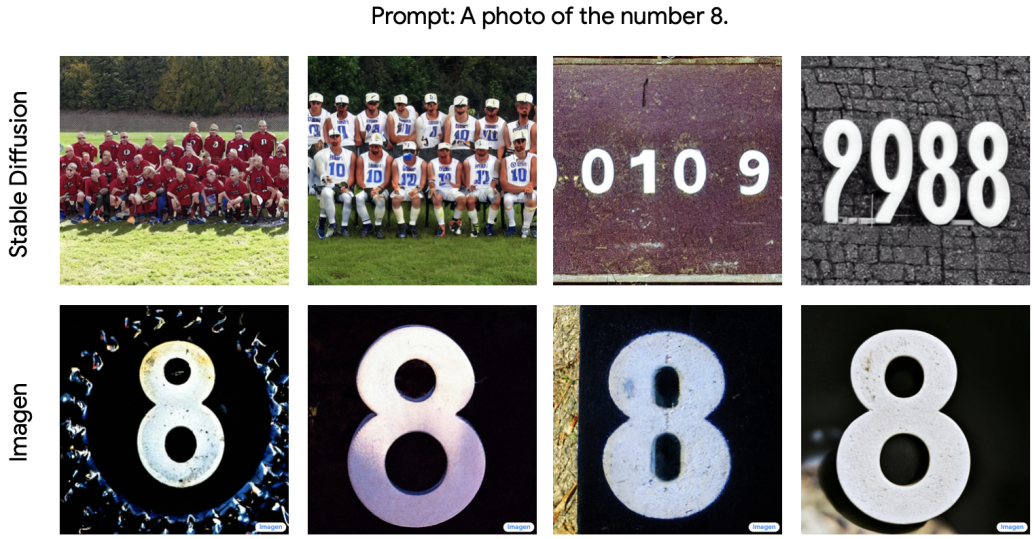

Figure 6: Example generated images from Stable Diffusion and Imagen for generating photos with text.

that upsamples from 256×256 to 1024×1024. However, we found only the 64×64 model to work well as a zero-shot classifier. The super-resolution models condition on a low-resolution input image, which means they denoise effectively regardless of the input prompt and thus aren't as sensitive to the class label. For example, unlike with Figure 3, high-resolution denoising with different text prompts produces images imperceptibly different to the human eye because they all agree with the same low resolution image. Imagen's super-resolution models are trained with varying amount of Gaussian noise added to the low-resolution input image (separate from the noise added to the high-resolution image being denoised). We were able to alleviate the above issue somewhat by using a large amount of such noise, but ultimately did not achieve very strong results with the high-resolution models. For example, the 64×64 to 256×256 model achieves an accuracy of 16.1% on ImageNet.

We further experimented with combining the low-resolution model's scores with the 64×64 to 256×256 model's. To do this, we used the learned weighting scheme detailed in Appendix B, but with learning 40 weights: 20 for the low resolution model and 20 for the super-resolution model. However, we found the learned weighting scheme put almost no weight on the super-resolution model's scores, and did not perform significantly better than the low-resolution model did on its own.