# A Additional Details

## A.1 Implementation Details

Here, we provide further architectural components in our implementation. This is an extension of Sec. 3.2.

*Injecting the Drag Encoding into DiT.* Recall that our flow encoder $F_l$ produces an encoding of the drags $\mathcal{D}$ at the latent resolution of DiT, *i.e.*, $F_l(\mathcal{D}) \in \mathbb{R}^{C_l \times h \times w}$. To inject this encoding to each DiT layer, the first step is to patchify $F_l(\mathcal{D})$ into a sequence of tokens, in line with how the noised latent is patchified. Then, we regress another set of token-wise scale and shift parameters $\alpha, \gamma$, and $\beta$ to add on top of those regressed from the diffusion time step.

*Injecting Global Image Control.* Following the text-to-image diffusion model, we also inject additional image control into the network by using the CLIP encoding of the input image $y$, which provides global information. Specifically, we leverage a pre-trained CLIP ViT/L-14 model to extract `[CLS]` tokens from every layer (25 in total) and train 25 separate linear layers on top of the features. The transformed features are then aggregated with average pooling and passed into another linear layer that produces the final image features. These features are subsequently injected into the SD UNet by replacing the CLIP textual encoding, or into DiT by replacing the class label (as in our case there are neither text prompts nor class labels).

## A.2 Experiment Details

*Training Details.* Our model was trained on 4× NVIDIA A40 GPUs with batch size $4 \times 32$ and learning rate $10^{-5}$ for $150,000$ iterations using `AdamW` optimizer. For the last $50,000$ iterations, we sample with probability 20% random texture renderings and 80% regular renderings in a batch. For each batch item, we also zero out the drags $\mathcal{D}$ 10% of the time to enable classifier-free guidance.

*Inference Details.* All our samples are generated using 50 denoising steps with classifier-free guidance weight 5. Generating an image roughly takes 2 seconds on a single NVIDIA A40 GPU.

*Evaluation Details.* For quantitative evaluations, we compute all metrics (PSNR, SSIM and LPIPS) at resolution $256 \times 256$. However, DragNUWA operates on a different aspect ratio (*i.e.*, $576 \times 320$). For a fair comparison, we first pad the square input image $y$ along the horizontal axis to the correct aspect ratio and resize it to $576 \times 320$, and then remove the padding of the generated last frame and resize it to $256 \times 256$ for evaluation. For methods that require a textual prompt (DragDiffusion, DragonDiffusion), we used "a rendering of a `[category]`" wherein `[category]` is obtained from GAPartNet's categorization. For InstructPix2Pix, we generated the best possible results with the following prompts: "pull the two drawers fully open" (Fig. 4a1), "turn the bucket handle down to the left" (Fig. 4b1), "pull the drawer directly beneath the table surface

outward" (Fig. 4c1), and "make every movable component articulate to its open position" (Fig. 4e1).

### A.3 Drag-a-Move Dataset Details

Here, we provide further details on the construction of our Drag-a-Move dataset. This is an extension of Sec. 4.

*Motivation Behind the Selection Criteria for 3D Models.* In GAPartNet [18], some object components exhibit minimal or no movement, such as the keys on a remote or the handle on a teapot, which are *not* useful for our objectives. Heuristically, we find parts that are annotated with "Hinge Handle", "Slider Drawer", "Hinge Lid" or "Hinge Door" display substantial motion. Hence, we discard objects that lack a minimum of one part annotated with these four kinematic types.

*Formulation of Asset Animation.* For each part $i$, we define an articulation state $A \in [0, 1]$, where $A = 0$ means fully closed and $A = 1$ means fully open. Because parts are essentially independent[5], we can sample any combination of states for the parts. We construct the data by creating short "animations" consisting of $N + 1 = 36$ frames. For each animation, we start by first sampling a random subset $s$ of parts that move during the animation, whereas the rest are stationary. Then, we generate two types of animations, selected with probability $p = 0.5$. For the first kind, the stationary parts $\bar{s}$ are fully closed, whereas the moving parts $s$ transition from fully closed to fully open, linearly. For the second kind, each stationary part $i \in \bar{s}$ is set to a randomly sampled state $c_i$, whereas each moving part $i \in s$ transitions from a random starting state $a_i$ to a random ending state $b_i$. Formally, the articulation state $A_{in}$ of moving part $i$ at time $n \in \{0, \dots, N\}$ is given by:

$$A_{in} = \begin{cases} \frac{n}{N} \mathbb{1}_{\{i \in s\}} & \text{if the animation is of type 1} \\ c_i \mathbb{1}_{\{i \notin s\}} + \frac{(N-n)a_i + nb_i}{N} \mathbb{1}_{\{i \in s\}} & \text{if the animation is of type 2} \end{cases} \tag{3}$$

where $a_i \sim U(0, \frac{1}{4}), b_i \sim U(\frac{3}{4}, 1)$ and $c_i \sim U(0, 1)$.

While in this formulation, each animation only contains parts opening, during training we randomly sample two frames, one as the reference image and the other as the ground truth image to be noised, irrespective of the order.

### A.4 Details on Drag-Driven Moving Part Segmentation

*Feature Extraction.* Following existing works on semantic segmentation using image generative models, the first step is to extract the diffusion model's internal features. Recall that at inference time, the model takes an image $y$ (to be segmented) and drags $\mathcal{D}$ as input and generates another image $x \sim \mathbb{P}(x|y, \mathcal{D})$

[5] With only a few exceptions where doors obstruct the movement of an internal part.

from pure Gaussian noise. Formally, given an input image-drags pair $(y, \mathcal{D})$, we first sample a noisy latent $z_t$ at time step $t$ as:

$$z_t = \sqrt{1-\sigma_t^2} z + \sigma_t \epsilon \tag{4}$$

where $z = E(y)$ is the latent code of $y$ and $\epsilon \sim \mathcal{N}(0, \mathbf{I})$ is the added noise. We encode $\mathcal{D}$ using our proposed flow encoder $F_l$ and extract the diffusion internal features $f_l$ for the pair $(y, \mathcal{D})$ by feeding it into the UNet at block $l$:

$$f_l = \text{UNet}(z_t, F_l(\mathcal{D})) - \text{UNet}(z_t, F_l(\phi)). \tag{5}$$

where $\phi$ represents no drags. Note we compute the feature *difference* here to better reflect the effect of the drags.

In our implementation, the drag conditions are injected into the self-attention layer of each UNet block. Empirically, we find using the output of the self-attention layer instead of the whole UNet block, provides better segmentation results. Following previous works [2,74], we resize the features at each block $l$ to the image resolution and concatenate them to create a feature pyramid.

*Clustering.* We perform K-means clustering on the feature pyramid to group the object's foreground pixels. We then obtain the segmentation by using the clusters to which the drag tails are associated.

We treat the time step $t$ used for feature extraction and the number of clusters $N_c$ as hyperparameters. An analysis of their effect is given in Appendix B.3.

# B Additional Results

## B.1 Additional Qualitative Results

Additional results on a variety of categories can be found in Fig. C. In instances where the conditional image features a complex, real-world background, the model adeptly "fills in" the occluded background details. This is notable given that its training exclusively utilizes images with white backgrounds. This capability is largely thanks to the strong priors of the pre-trained SD model.

## B.2 Failure Cases

We show typical failure cases of our model in Fig. A. Notably, the model's limitations, potentially attributed to the limited scale of the training data, manifest in several ways: it may i) incorrectly recognize different motion parts of an object (left); ii) unrealistically distort an object from an *unseen* category (middle); or iii) occasionally create images that lack physical plausibility (right).

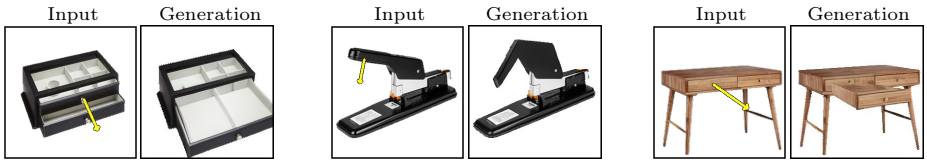


**Fig. A: Failure Cases.** While our model is capable of generating images of complex part-level dynamics, it occasionally produces physically implausible samples.

| # of Clusters | DragAPart Features | | | | | Baseline Features | | |
|---|---|---|---|---|---|---|---|---|
| $N_c$ | $t=0$ | $t=200$ | $t=500$ | $t=800$ | $t=999$ | CLIP | DINO | SD |
| 2 | 22.79 | 22.71 | 21.72 | 23.32 | **23.39** | 13.89 | 21.24 | 14.47 |
| 3 | 26.84 | **27.22** | 26.94 | 26.42 | 25.81 | 14.00 | 24.13 | 14.38 |
| 4 | 26.06 | **27.29** | 26.45 | 26.62 | 26.40 | 14.49 | 24.16 | 14.68 |
| 5 | 25.73 | 25.48 | 25.21 | 26.02 | **26.10** | 13.55 | 24.58 | 14.62 |

**Table A: Quantitative Evaluation of Drag-Driven Moving Part Segmentation.** Clustering of our DragAPart features yields the best (*i.e.*, highest) mIoU, and is relatively robust across different hyperparameter settings.

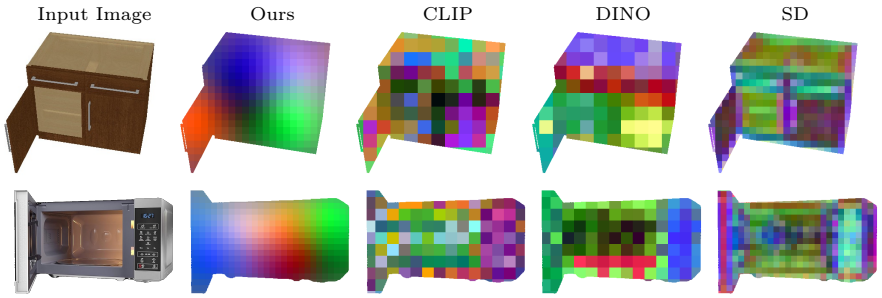


**Fig. B: Visualization of Image Features.** Notably, the features extracted through our DragAPart are smooth and exhibit part-level information.

### B.3 Quantitative Comparisons of Moving Part Segmentation

We assess our proposed moving part segmentation approach on the test split of Drag-a-Move, where part-level segmentation is annotated so the ground-truth masks are available. In Tab. A, we provide the average mask Intersection over Union (mIoU) obtained using our method and baselines of clustering alternative CLIP [49], DINO [9,43] and SD[6] [52] features. We also visualize the corresponding features (transformed through principal component analysis on foreground pixels to 3 channels as RGB values) in Fig. B. The results underscore that the features extracted through our DragAPart demonstrate better part-level information.

[6] To ensure fairness, the metric for SD is obtained by selecting the maximum from the 5 mIoU values with diffusion time step set to 0, 200, 500, 800 and 999 respectively.

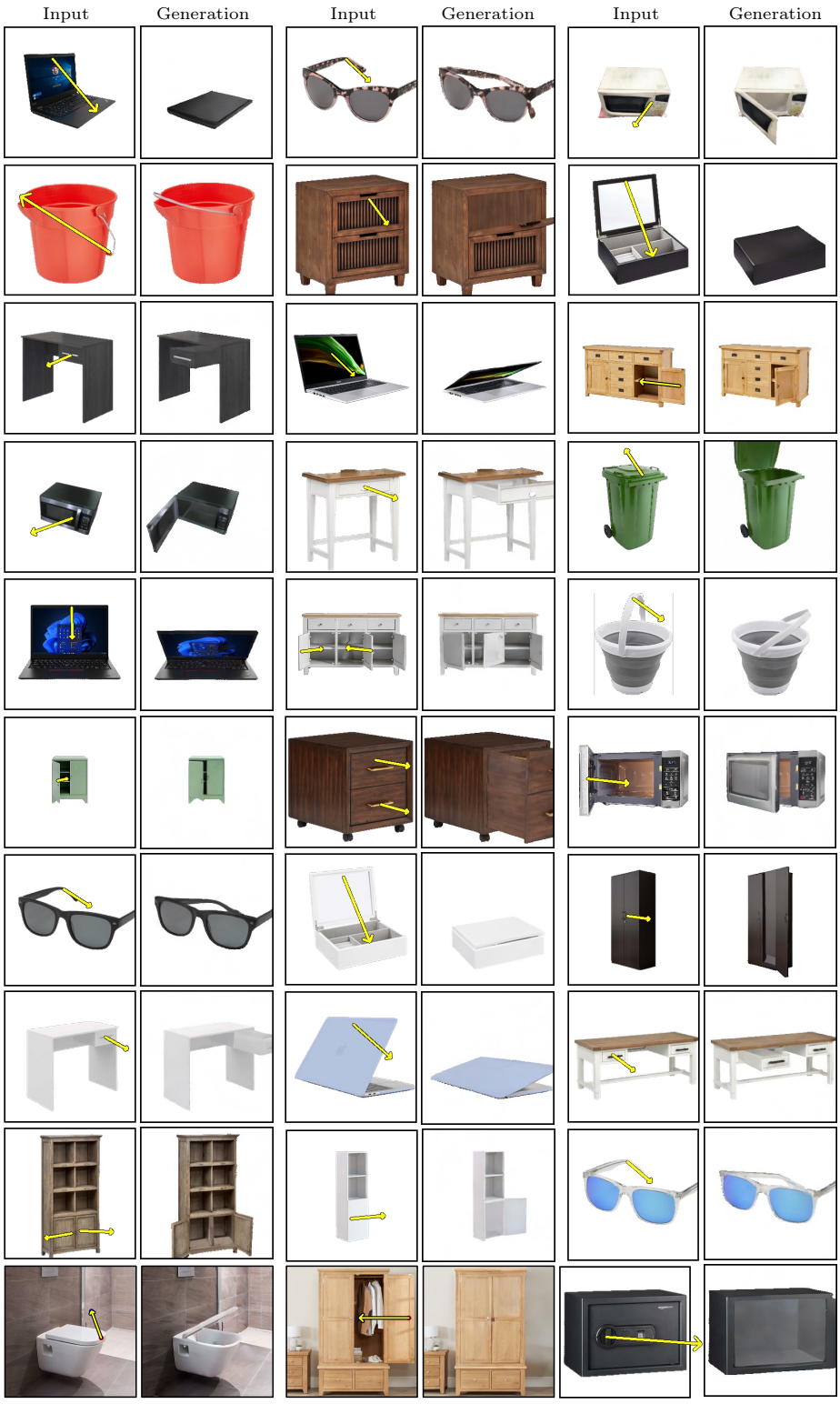


**Fig. C: Additional Qualitative Results.**