UPLiFT: Efficient Pixel-Dense Feature Upsampling with Local Attenders (Supplemental Material)

Matthew Walmer[1] Saksham Suri[2] Anirud Aggarwal[1] Abhinav Shrivastava[1]
[1]University of Maryland, College Park [2]Meta

A. Additional Details for Predictive Tasks

A.1. Baseline Methods

In this work, we focus on comparing UPLiFT with other task-agnostic feature upsamplers, which have grown in popularity in recent years. Such methods take existing pre-trained feature extractors, and provide an upsampler add-on to provide dense, powerful features directly out of the box. We compare with the following methods:

FeatUp [4]: We compare with the JBU FeatUp variant, which is an iterative upsampler that performs $16\times$ upsampling with a stack of four modified Joint Bilinear Upsamplers [9]. While the implicit variant of FeatUp produces impressive results, it has been shown by works like [6] that this approach is too slow to be practical for large-scale evaluations, with functional inference speeds over $500\times$ slower than comparable methods. We use the official FeatUp JBU model distributed by [4] trained for DINOv2-S/14 without backbone normalization.

LiFT [16]: LiFT is also an iterative upsampling method, which in its base form performs $2\times$ upsampling. The same LiFT model can also be applied iteratively four times to perform $16\times$ upsampling. However, [2] has shown that this iterative upsampling can lead to semantic drift and degraded features. For this reason, we follow the example of [2] and present LiFT in two configurations: "LiFT" runs the model four times for $16\times$ upsampling and "LiFT-$2\times$" runs the model only once, then follows this with bilinear upsampling to pixel scale. As there is not an official LiFT model for DINOv2-S/14, we train one for this work.

LoftUp [6]: LoftUp uses cross-attention to directly upsample low-resolution backbone features to pixel-scale features, with a $14\times$ upsampling step. We compare with the official LoftUp for DINOv2-S/14 model distributed by [6]. We note that this LoftUp model was originally trained with the DINOv2-S/14 backbone distributed by [11], while other recent methods (ours included) use the version distributed by Hugging Face[1]. While these DINOv2 models are fundamentally the same model trained on the LVD-142M dataset and should produce similar results, we choose to conduct an additional test to determine if this has any impact on the performance of LoftUp. We perform evaluation twice with LoftUp, once with each version of DINOv2-S/14, and report the results in Table 1. We find that LoftUp's performance is nearly identical for both backbones, with only minor variations seen in the mIoU scores for VOC and Cityscapes. For this reason, we conclude that these minor variations in backbone distributions are not a significant confounding factor in our results, and we thus choose to present results for LoftUp with its original backbone for sake of fair representation.

JAFAR [2]: JAFAR also uses a cross-attention-based approach to perform direct $14\times$ feature upsampling. We build on the evaluation protocols of JAFAR for segmentation and depth estimation, and we compare with the official JAFAR for DINOv2-S/14 model from [2].

AnyUp [18]: AnyUp proposes a modified version of the JAFAR architecture which is designed with an additional "feature-agnostic layer" that enables the model to generalize to different backbones at inference time. Note that AnyUp is still initially trained with features from a particular backbone for upsampling training. Moreover, the results of [18] show that performing inference with a different backbone than was used in training leads to poorer downstream performance than using the same backbone for both training and evaluation, suggesting that training model-specific upsamplers is still preferable for performance. For this study, we compare with the official AnyUp model distributed by [18].

A.2. Upsampling Rates

For our main evaluation, we use DINOv2-S/14 as the standard backbone, and run all methods in a configuration to produce pixel-dense features, which requires $14\times$ upsampling. For methods that perform $16\times$ upsampling, which includes our UPLiFT, we follow the example of [2] and simply over-upsample the features, and then downsample to pixel-resolution. We note that this puts UPLiFT at a

[1] https://huggingface.co/timm/vit_small_patch14_dinov2.lvd142m

Table 1. **Evaluating the impact of minor backbone variations.** We present a comparison of baseline LoftUp [6] running either with its original DINOv2-S/14 backbone from [11], or with an alternate distribution of DINOv2-S/14 provided by Hugging Face, which is commonly used in other upsampling works. Note that both models are theoretically the same model and only represent different distribution platforms. We find that there is minimal variation in the resulting performance, with the largest changes being in VOC mIoU and Cityscapes mIoU. Our UPLiFT surpasses LoftUp's performance for either backbone. We report results for LoftUp with its original backbone in the main work to provide a fair representation of the method.

Upsampler			Semantic Segmentation								Depth Estimation	
			COCO		VOC		ADE20K		Cityscapes		COCO	
Method	Params (M)	Time (ms)	mIoU ↑	Acc ↑	mIoU ↑	Acc ↑	mIoU ↑	Acc ↑	mIoU ↑	Acc ↑	δ_1 ↑	RMSE ↓
LoftUp + Orig Backbone	4.3	223.5	62.19	81.35	84.63	96.33	42.16	75.79	62.09	93.55	58.69	0.68
LoftUp + HF Backbone	4.3	223.5	62.20	81.35	84.52	96.30	42.16	75.77	62.17	93.56	58.70	0.68
UPLiFT	0.8	79.4	**62.55**	**81.57**	**85.21**	**96.51**	**42.97**	**76.00**	**65.38**	**94.41**	**61.16**	**0.63**

computational efficiency disadvantage compared to cross-attention-based methods, but despite this, we still achieve faster inference than the cross-attention baselines. Moreover, we note that more recent self-supervised backbones, like DINOv3 [15], have moved back to using patch size 16, which makes $16\times$ upsampling the likely desired option for future research.

A.3. Training Cost Comparison

We briefly compare the training cost of UPLiFT to the recent state-of-the-art methods JAFAR [2], AnyUp [18], and LoftUp [6]. As discussed in Section 3.3, we train UPLiFT for one epoch on the ImageNet-1K dataset [3] using a multi-scale and multi-step training configuration. Under our final protocol, UPLiFT training with DINOv2-S/14 takes $\sim$9 hours on one A5000 GPU. For comparison, when using the official training code of [2], we find that JAFAR training takes $\sim$3 hours on one A5000. [18] reports that AnyUp training takes $\sim$5 hours on one H100 GPU. The training protocols of AnyUp are based on those of JAFAR, though they use additional data augmentation with four random crops per image. While UPLiFT training does take a few hours longer than JAFAR and AnyUp, we believe all three are reasonably comparable as they can all be trained overnight with one moderate-strength GPU. We also note that LoftUp training comes with a comparatively higher computational price. The official training code of [6] recommends training with 4 GPUs, and when using 4 A5000s, we estimate that LoftUp would take $\sim$50 hours to train on a 1M image subset of SA1B [8] as recommended in [6]. Note that this estimate is only for the first training phase, and the additional self-distillation phase will add further training time. Overall, the training cost for UPLiFT is comparable to JAFAR and AnyUp, and far lower than LoftUp.

A.4. Local Attender Details

To provide additional details on our implementation of Local Attender, we present pseudocode in Algorithm 1. For sake of simplicity, we exclude the batch dimension from all notation. The primary operations used in Local Attender are matrix offsetting and element-wise matrix multiplication with broadcasting over placeholder dimensions. In broad terms, the steps are as follows. First, a 1×1 convolution is applied to Guide feature map G to set its channel depth to n, the size of the Neighborhood. Softmax is applied to create the Attender Map, A. G (and thus A) is larger than V by an integer factor c in the spatial dimensions, so we reshape A to divide it into $c\times c$ cells. Then, the Value feature map V is also processed. Replication Padding is applied, and we then generate a set of offset feature maps, V_{off}, based on the Neighborhood positions. Element-wise matrix multiplication with broadcasting is then performed between A and V_{off}, and the attention-weighted features are then summed. Finally, we concatenate the cells to produce the final upsampled feature map $\widetilde{V}$.

B. Additional Details for Generative Tasks

B.1. UPLiFT for VAE Features

We describe additional details of our UPLiFT model designed for VAE feature upsampling and generative tasks.

UPLiFT size for VAE. We empirically find that small parameter count upsampling models, like those demonstrated to be effective for predictive downstream tasks, are insufficient for generative downstream tasks. We illustrate this in Figure 1, where we present a comparison with a small UPLiFT model, which has roughly $2.8M$ parameters and is trained using the same training protocols as our main model. In this comparison, we see that the resulting model produces blurry, low quality upsampling, and is not able to capture high-frequency information. We theorize that the necessary model capacity to train an effective feature upsampler for generative tasks is intrinsically larger than the necessary size for predictive tasks. We base this theory on the intuition that predictive tasks are about narrowing down information into a compressed understanding, which can be represented more compactly with a smaller network, while generative tasks require a high level of detail over a diverse

Algorithm 1 $\widetilde{V} \leftarrow \text{LocalAttender}(V, G|W, N)$

LocalAttender upsamples "Value" feature map V using a high-res "Guide" feature G to apply local attention over Neighborhood N, which is defined as a collection of fixed integer offsets in 2D space.

Inputs: $V \in \mathbb{R}^{C_V \times H \times W}$ and $G \in \mathbb{R}^{C_G \times cH \times cW}$ where $c \in \mathbb{Z}$
Outputs: $\widetilde{V} \in \mathbb{R}^{C_V \times cH \times cW}$
Parameters: $W \in \mathbb{R}^{C_G \times n}$ where $n = ||N||$
Hyperparameters: $N = o_1, o_2, ..., o_n$ where $o_i \in \mathbb{Z}^2$

$A \leftarrow \text{Conv}_{1\times1}(G, W)$ ▷ Attender Map A now has shape $[n, cH, cW]$
$A \leftarrow \text{Softmax}(A, \text{dim} = 1)$
$A \leftarrow \text{DivideCells}(A, \text{size} = (c, c))$ ▷ A now has shape $[n, H, c, W, c]$
$A \leftarrow \text{ExpandDims}(A, \text{dims} = 1)$ ▷ A now has shape $[1, n, H, c, W, c]$
$V_{\text{pad}} \leftarrow \text{ReplicationPad2d}(V)$
$V_{\text{off}} \leftarrow \text{Stack}(\text{ApplyOffset}(V_{\text{pad}}, o_i)\ \forall o_i \in N)$ ▷ V_{off} now has shape $[C_V, n, H, W]$
$V_{\text{off}} \leftarrow \text{ExpandDims}(V_{\text{off}}, \text{dims} = (4, 6))$ ▷ V_{off} now has shape $[C_V, n, H, 1, W, 1]$
$\widetilde{V} \leftarrow \text{BroadcastMultiply}(A, V_{\text{off}})$ ▷ $\widetilde{V}$ now has shape $[C_V, n, H, c, W, c]$
$\widetilde{V} \leftarrow \text{Sum}(\widetilde{V}, \text{dim} = 2)$ ▷ $\widetilde{V}$ now has shape $[C_V, H, c, W, c]$
$\widetilde{V} \leftarrow \text{ConcatCells}(\widetilde{V})$ ▷ $\widetilde{V}$ now has shape $[C_V, cH, cW]$

range of visual textures and patterns to achieve high quality. For this reason, we train a larger UPLiFT model for generative tasks, increasing the number of layers in both the encoder and decoder module and increasing the channels per layer. This larger size UPLiFT has $53.5M$ parameters, which still makes it significantly smaller than the compared CFM [14] models, which have $113M$ or $306M$ parameters.

Refiner Block. We find that it is beneficial to introduce an additional "Refiner Block" after the Local Attender module, which is designed to realign the output features with the distribution expected by the VAE decoder. We demonstrate the importance of the refiner block by ablating it in Figure 1 (Right). Without the refiner block, the upsampled images have significant blocky artifacts, which are likely the result of the strict, linear-combination feature upsampling approach. We note that our Local Attender module, and the cross-attention modules used by recent works, act as a form of strong feature regularization, which ensures that the features output by the upsampler maintain a similar distribution to the original input distribution, which, in theory should also match the distribution expected by the VAE decoder. However, through testing, we find that this design may be too restrictive for a generative context. For this reason, we introduce a post-attender "Refiner Block" which gives UPLiFT an opportunity to improve the final features.

Noise Layers and Augmentation. We follow the example of [14] and concatenate additional gaussian random noise channel inputs at several points in the UPLiFT Encoder module when upsampling VAE features. These additional noise channels are provided to help seed the generation of high resolution details. We also apply gaussian noise augmentation to the input latent representation, following [14].

Color Correction. When training UPLiFT for VAE, all our loss terms are computed in the latent feature space, which improves the efficiency of training by removing the need to decode high-resolution images. Our UPLiFT training is effective at minimizing the $L2$ distance of upsampled features in latent space, however, we find that small perturbations in latent space can lead to slight color shifts after decoding. While this could likely be addressed through the use of pixel-space loss terms, this would greatly increase training costs. Instead, we introduce a simple color correction module after the VAE Decoder at inference time. This module computes the per-color-channel means for the low-resolution input image and the high-resolution output image, and subtracts the difference vector from the output image to realign the color mean. We find that this simple module is sufficient to remove any minor color shifts.

Layer Normalization. Finally, for our larger VAE UPLiFT model, we find it is beneficial to replace the Batch Norm [7] layers with Layer Norm [1] instead, as it provides better numerical stability for the deeper architecture. When using Batch Norm, we sometimes observe color blob artifacts in the upsampled images, which are a consequence of numerical instability in the model. After replacing the Batch Norm operations with Layer Norm, these artifacts no longer occur.

B.2. Experimental Methods for Generative Tasks

We summarize additional key details of our experimental protocols for generative tasks.

Diffusion Upsampling. We evaluate UPLiFT's ability to upscale diffusion features on COCO 2014 and reLAION-400M following the protocols of [14]. On COCO, we randomly sample 5k caption-image pairs. On reLAION, we randomly sample 5k images with a minimum resolution of

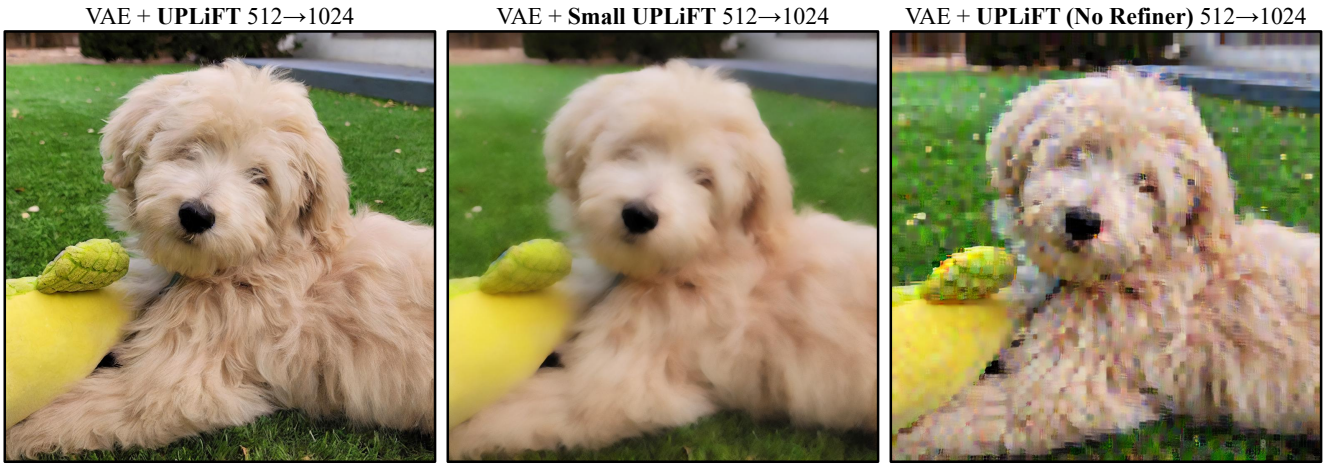

Figure 1. **Visual ablation of design choices for VAE UPLiFT.** (Left) UPLiFT achieves high quality 512→1024 upsampling with a larger parameter count model. (Middle) A smaller-scale UPLiFT has insufficient capacity to upsample all high-frequency information and produces blurry results. (Right) Ablation of the Refiner Block leads to blocky artifacts in upsampled images.

1024×1024. While [14] used the now deprecated LAION dataset, our sample is sufficiently similar, and our computed FID with LCM-LoRA SDXL matches that of CFM. During inference, we generate latents with Stable Diffusion 1.5 [13] for each sampled caption, and then decode them to generate 5k 512×512 images. We apply UPLiFT to produce 1024×1024 images. Note that UPLiFT leverages the image created from the decoded low-resolution latents to guide feature upsampling, similar to CFM, which must decode the image as part of the Pixel-Space Upsampling (PSU) method. However, UPLiFT does not require re-encoding a bilinearly upsampled image like CFM does, which gives an additional efficiency gain to UPLiFT.

The default precision of `float32` is used for all diffusion processes, and each model utilizes its default scheduler with the specified number of steps. We also compare with SDXL [12] and LCM-LoRA SDXL [10], which natively generate at 1024×1024, to provide an additional point of reference for visual quality from high-resolution latents. SDXL can be viewed as an 'upper-bound' oracle for native megapixel generation, while LCM-LoRA SDXL provides a point of reference of a low-latency distilled model at megapixel native generation. Please note that SD 1.5 cannot natively generate latents for images at resolutions different from 512×512 without significant quality degradation or alterations. [14] includes experiments upscaling 256×256 to 1024×1024 by specially fine-tuning a Stable Diffusion 1.5 model at a lower resolution. For our tests, we choose to prioritize experiments using only the official Stable Diffusion 1.5 model generating latents for 512×512 images. Utilizing the official code of [14], we compute FID and patch-FID, which splits the image into four random patches sized 512×512. For CLIP score, we use the Python package `clip_score` with `clip-vit-base-patch32`.

Super-Resolution. Following CFM [14], we perform super-resolution evaluations on two datasets: FacesHQ and LHQ. We randomly sample 5k images from the LHQ dataset and the joint combination of CelebA-HQ and FFHQ datasets, called FacesHQ. For each image, we perform a 1024×1024 center crop, followed by a bilinear downsample to 256×256. Finally, we apply UPLiFT to recover the full sized image. To do so, we first use Stable Diffusion 1.5's VAE to encode the image into latent space. Then we apply the UPLiFT module with 2 iterations, resulting in a $4\times$ latent, which is decoded using the same VAE's decoder back into pixel space. Performance is measured using the evaluation scripts of [14], which track SSIM [17], PSNR, FID, and patch-FID.

Latency Timing. To ensure fair comparison with [14], we measure latency on a single NVIDIA A100-SXM4-80GB across both experiments. We use a batch size of 1 for experiments on reLAION, and 4 for experiments on COCO to be consistent with [14]. All experiments utilize the `torch.compile` module, and measurements are conducted with 3 warm-up batches. The final latency is averaged across 10 subsequent batches.

C. Ablations of UPLiFT Design Choices

C.1. Training Depth

As discussed in Section 3.3, we train UPLiFT with a multi-depth, multi-step feature reconstruction training objective, where the level of downsampling and the number of upsampling steps depends on the training depth, d. Here we present an ablation over d, where we train our UPLiFT model with different training depth configurations, including individual depths, or multiple concurrent depths. We use semantic segmentation on COCO and VOC with pixel-

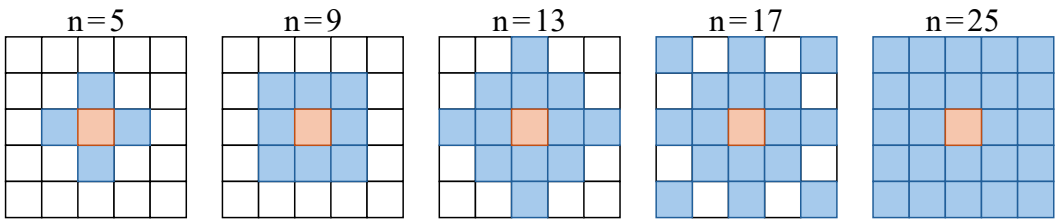

Figure 2. **Local Attender neighborhood designs.** We visualize the neighborhood designs tested with our Local Attender module. The center token (orange) is always included in the neighborhood, and the blue tokens represent the offset relative local positions that are included in feature pooling through local attention.

Table 2. **Ablation of training depth configurations.** We test COCO and VOC segmentation for different training depth levels, including individual and concurrent depth configurations.

	COCO		VOC	
Depth	mIoU ↑	Acc ↑	mIoU ↑	Acc ↑
$d = 1$	62.26	81.34	85.03	96.42
$d = 2$	62.34	81.43	84.89	96.40
$d = 3$	62.24	81.34	84.67	96.33
$d = 4$	60.82	80.30	81.42	95.42
$d \in \{1, 2\}$	<u>62.46</u>	<u>81.50</u>	<u>85.04</u>	<u>96.44</u>
$d \in \{1, 2, 3\}$	**62.55**	**81.57**	**85.21**	**96.51**
$d \in \{1, 2, 3, 4\}$	62.36	81.45	84.97	96.42

Table 3. **Ablation of Local Attender neighborhood sizes.** We compare UPLiFT performance with Local Attender modules with different neighborhood patterns, or without the Local Attender.

	COCO		VOC	
Neighbors	mIoU ↑	Acc ↑	mIoU ↑	Acc ↑
No LA	47.52	73.06	64.86	91.37
$n = 5$	62.15	81.30	84.80	84.80
$n = 9$	62.33	81.42	84.99	96.44
$n = 13$	<u>62.51</u>	<u>81.53</u>	85.10	96.47
$n = 17$	**62.55**	**81.57**	<u>85.21</u>	**96.51**
$n = 25$	62.46	81.50	**85.22**	<u>96.48</u>

dense features as our evaluation tasks for this ablation. The results are summarized in Table 2.

When training with only one depth level (rows $1{-}4$), better performance is achieved with shallower depths. VOC has the best performance with $d{=}1$, and COCO has slightly better performance with $d{=}2$. We believe this occurs because shallower depths give the UPLiFT Encoder a large input image, which allows it to learn more about high-frequency image information. For $d{=}3$, we see a slight drop in performance and for $d{=}4$ we see a major drop in performance. Under $d{=}4$, the $448{\times}448$ image is downsampled to $24{\times}24$ which yields only a $2{\times}2$ token grid. It seems this level of downsampling is too severe for effective learning.

Next, we find that training with multiple depths concurrently can enhance performance. As shown by row 5, training with $d \in \{1, 2\}$ concurrently leads to superior performance on both COCO and VOC. This effect is further enhanced by adding in $d{=}3$. However, the addition of $d{=}4$ becomes detrimental for multi-depth training, which again indicates that downsampling the input too severely hinders learning. From this, we find that training with a multi-depth configuration with $d \in \{1, 2, 3\}$ yeilds the best results, so we use this training configuration in our predictive task experiments. Note that for our VAE UPLiFT model, we train with a maximum resolution of $1024{\times}1024$ in following [14]. For this larger image size, we find that $d \in \{1, 2, 3, 4\}$ works to good effect.

C.2. Local Attender Neighborhood

We propose a new Local Attender module, which uses a relatively-defined, variably-sized local neighborhood to perform local attention pooling in linear time. In this section, we present additional experiments with different neighborhood sizes and shapes. We limit the neighborhood size to a maximum of a $5{\times}5$. We present an illustration of the neighborhood sizes and shapes tested in Figure 2. In addition, we present results for an UPLiFT model with the Local Attender module ablated. All models are trained for one epoch on ImageNet-1k, with $448{\times}448$ max image size, like the main work. Results are summarized in Table 3.

First, from row 1, we see that the removal of the Local Attender module significantly harms performance, which indicates that our Local Attender is essential for maintaining effective feature upsampling through multiple steps. Next, in row 2, we test $n{=}5$ with what we consider to be the smallest viable neighborhood, which consists of only the current token and its immediately touching neighbors. Already for $n{=}5$ we see strong performance in both datasets, though this performance continues to improve as we expand the neighborhood to 9, 13, and 17 neighbors. The $n{=}25$ configuration, which compared to $n{=}17$ adds on several additional neighbors along the second layer of offsets, only leads to roughly equal performance in VOC and slightly lower performance in COCO. This result indicates that the

Table 4. **Ablation of Encoder and Decoder Size.** We compare our final UPLiFT design to variations with smaller or larger encoders and decoders or alternately no encoder module.

	Upsampler		COCO		VOC	
Model	Params (M)	Time (ms)	mIoU ↑	Acc ↑	mIoU ↑	Acc ↑
Enc_{None}	0.36	64.6	61.81	81.02	84.19	96.21
Enc_{Small}	0.50	71.4	62.48	81.51	85.21	96.46
Enc_{Large}	1.79	98.7	62.51	81.54	**85.22**	96.48
Dec_{Small}	0.50	75.9	62.37	81.43	84.91	96.41
Dec_{Large}	1.57	88.3	**62.58**	**81.60**	**85.22**	96.48
UPLiFT	0.79	79.4	62.55	81.57	85.21	**96.51**

Table 5. **Local Attender *vs*. Windowed Cross Attention.** We present an ablation using Windowed Cross Attention (WCA) in place of our Local Attender (LA) module. Local Attender results in better performance and efficiency over WCA.

		Upsampler		COCO		VOC	
Att.	Heads	Params (M)	Time (ms)	mIoU ↑	Acc ↑	mIoU ↑	Acc ↑
WCA	1	0.96	87.4	61.52	80.92	84.06	96.15
WCA	4	0.96	90.0	61.59	81.04	84.12	96.21
WCA	8	0.96	91.6	61.53	80.99	84.25	96.19
WCA	12	0.96	93.8	61.35	80.84	83.98	96.13
LA	-	0.79	79.4	**62.55**	**81.57**	**85.21**	**96.51**

Table 6. **UPLiFT with Additional Non-ViT Backbones.** We demonstrate the effectiveness of UPLiFT with additional convolution-based backbone's features.

Features		COCO		VOC	
Backbone	Upsampler	mIoU ↑	Acc ↑	mIoU ↑	Acc ↑
ResNet-50	Bilinear	40.26	65.28	49.87	85.72
ResNet-50	UPLiFT	**43.90**	**68.81**	**54.75**	**86.94**
ConvNeXt-S	Bilinear	45.76	68.15	57.71	87.14
ConvNeXt-S	UPLiFT	**47.14**	**69.37**	**59.48**	**87.48**

addition of further neighbors may not be beneficial if they are added in intermediate places between existing neighbors. Overall, our best performance is achieved with the star-shaped n=17 neighborhood design, so we use this as our Local Attender neighborhood for all our experiments in the main work.

C.3. UPLiFT Encoder and Decoder Designs

In this section, we provide additional details of our UPLiFT Encoder and Decoder modules, and we present ablations testing larger and smaller variations of each module. Our UPLiFT Decoder module does the primary work of upsampling the low resolution backbone features, while the UPLiFT Encoder extracts shallow, high-res features to help guide feature upsampling. Unlike LiFT, which runs its encoder module iteratively after each upsampling step, we use a single-encoder-pass approach, where the encoder module is only run once with the original resolution input image. Our encoder maintains the same spatial resolution as the input image, and nearest neighbor downsampling is applied to the resulting features before guiding each upsampling step. Our Encoder and Decoder modules are both lightweight, convolutional networks, with 10 and 6 conv layers respectively, along with batchnorm and ReLU operations. The Decoder also includes a deconv layer to increase the feature resolution $\times 2$. Our official UPLiFT repository (https://github.com/mwalmer-umd/UPLiFT/) includes code and configuration files with full details for both of these modules.

We further demonstrate the optimality of our final UPLiFT design by comparing it with variations with larger and smaller encoders and decoders. These variations are creating by halving or doubling the channel depth of each convolutional layer. We additionally present an ablation where the encoder module is removed completely. As shown in Table 4, we find that the smaller modules underperform, while the larger ones increase cost (doubled parameters/slower speed) with minimal performance gain. In addition, fully ablating the encoder module significantly harms COCO and VOC performance. Thus, our proposed UPLiFT design offers the best performance-efficiency trade-off compared to the other configurations.

C.4. Comparison with Windowed Self-Attention

We present an ablation testing the effectiveness of our Local Attender module in comparison with Windowed Cross-Attention (WCA), which may also be referred to as Neighborhood Attention [5]. Like Local Attender, WCA can provide a mechanism to pool local features to produce an upsampled feature map that adheres to the distribution of the original backbone map. The concurrent work AnyUp leverages WCA in this way. When implemented efficiently, WCA can also avoid the quadratic time scaling of full cross-attention. However, we believe our Local Attender has an additional advantage: Local Attender re-formulates the local attentional operations using consistent, relatively-defined local-offsets, which we theorize provides stronger inductive biases to guide learning. To demonstrate this, we train several UPLiFT models which use WCA in place of Local Attender. We use the efficient implementation of NATTEN [5] with a window size of 5×5 to provide a level comparison. We test WCA with several attention head counts, ranging from 1 to 12. From our result in Table 5, we find that WCA with one head has significantly lower performance. Running WCA with 4 or 8 heads achieves performance more comparable to the baseline method JAFAR, while 12 heads is actually detrimental. However, our UPLiFT with Local Attender has better performance in both metrics and datasets over all variations using WCA. Overall, these results demonstrate the potential strengths of our Local Attender design over Windowed Cross-Attention for use in feature upsampling.

Table 7. **Semantic Segmentation with DINOv3 and UPLiFT.** We measure segmentation performance on COCO and VOC for DINOv2-S/14 *vs*. DINOv3-S+/16. We compare against JAFAR and AnyUp, which either report results with this DINOv3 backbone or have provided an official model for it [2]. We find UPLiFT gives the best performance for all metrics, datasets, and backbones. †Row adapted from [18].

Upsampler		DINOv2-S/14				DINOv3-S+/16			
		COCO		VOC		COCO		VOC	
Method	Params (M)	mIoU ↑	Acc ↑	mIoU ↑	Acc ↑	mIoU ↑	Acc ↑	mIoU ↑	Acc ↑
JAFAR	0.7	61.71	81.01	<u>84.38</u>	<u>96.22</u>	62.47	81.50	<u>83.05</u>	<u>95.99</u>
AnyUp†	0.9	<u>62.16</u>	<u>81.37</u>	84.00	96.19	<u>62.99</u>	<u>81.84</u>	–	–
UPLiFT	0.8	**62.55**	**81.57**	**85.21**	**96.51**	**63.74**	**82.27**	**84.72**	**96.55**

D. Additional Results and Visualizations

D.1. UPLiFT for Convolutional Backbones

To demonstrate the generality of UPLiFT, Table 6 presents results for two additional non-ViT timm backbones with different training protocols: ResNet-50 with YFCC100M ssl-pretraining & IN1k finetuning, and ConvNeXt-S with IN22k pretraining & IN1k finetuning. We use bilinear feature upsampling as a baseline. UPLiFT yields better performance for both datasets and metrics. We would also note that the SD1.5 VAE backbone used in Section 5 is primarily convolution-based. Overall, these results demonstrate that our UPLiFT approach is effective for both ViT-based and convolution-based backbones and features.

D.2. UPLiFT for DINOv3

In our primary results in Section 4, we follow the example of recent feature upsampling works and use DINOv2-S/14 [11] as our primary backbone for assessing UPLiFT and comparing it with baseline works. However, as of writing, DINOv3 [15] has recently been published, and we expect future work in feature upsampling will likely shift to focus on this family of backbones. For this reason, we present additional results for Semantic Segmentation using DINOv3-S+/16. We compare with JAFAR [2], which has released an official model for this backbone on their repository, and with AnyUp [18], which has published results for DINOv3 on COCO in their work. While we would additionally like to compare with LoftUp [6], no official DINOv3 LoftUp upsampler has been published as of writing, nor has official training code been made publicly available. We present results for COCO and VOC in Table 7, alongside results for DINOv2-S/14 for comparison. We see that the DINOv3 features lead to stronger performance in COCO, with improved results for DINOv3 with UPLiFT. However, DINOv3 does not necessarily yield stronger performance in VOC, as is seen for both UPLiFT and JAFAR. However, for both the DINOv2 and DINOv3 backbones, UPLiFT achieves the best semantic segmentation performance for both metrics and datasets.

D.3. Qualitative Comparisons with Baselines

Following the example of prior works [2, 4, 6, 18], we present a qualitative comparison of our upsampled features with baseline methods using Principle Component Analysis (PCA). As shown in Figure 3, UPLiFT and the other recent methods are effective at maintaining semantic consistency for key objects, as indicated by consistent coloration with the base features. LiFT's upsampled features show signs of visual degradation, which we examine more in Section D.5. Qualitatively, UPLiFT's features are comparable to the recent cross-attention-based methods JAFAR, AnyUp, and LoftUp. We note that UPLiFT features can sometimes appear smoother, which is likely a result of our iterative feature growing method. In some cases, JAFAR, AnyUp, and LoftUp appear sharper, but this sharpness often comes with noisier edges. This can also lead to feature-bleed, where features from unrelated background regions are found in foreground objects. This effect can be seen in row 2 for JAFAR and AnyUp and in row 3 for LoftUp. UPLiFT's focus on iterative feature growing with local neighborhoods tends to produce more locally consistent features, which is likely beneficial for downstream tasks like segmentation.

We also note that DINOv2's underlying features tend to have noisy artifacts in smooth background regions, like the sky, as shown in row 2. While most methods preserve these artifacts in some form after upsampling, LoftUp is the only method that actively removes them. This is a result of LoftUp's training protocol, which uses additional pseudo-supervision from SAM [8] to refine and remove these artifacts from the training data. While the removal of these artifacts may look better qualitatively, it is difficult to judge if this is practically beneficial, as it does cause LoftUp to deviate more significantly from the original backbone's feature distribution. In addition, we note that the recently released DINOv3 [15] exhibits fewer background artifacts compared to DINOv2, so actively removing artifacts from training data may be a lower priority for future models. In summary, these PCA visualizations help to reveal differing properties of the features created by different upsampling methods, but we believe these assessments should also be considered alongside quantitative evaluations in downstream tasks.

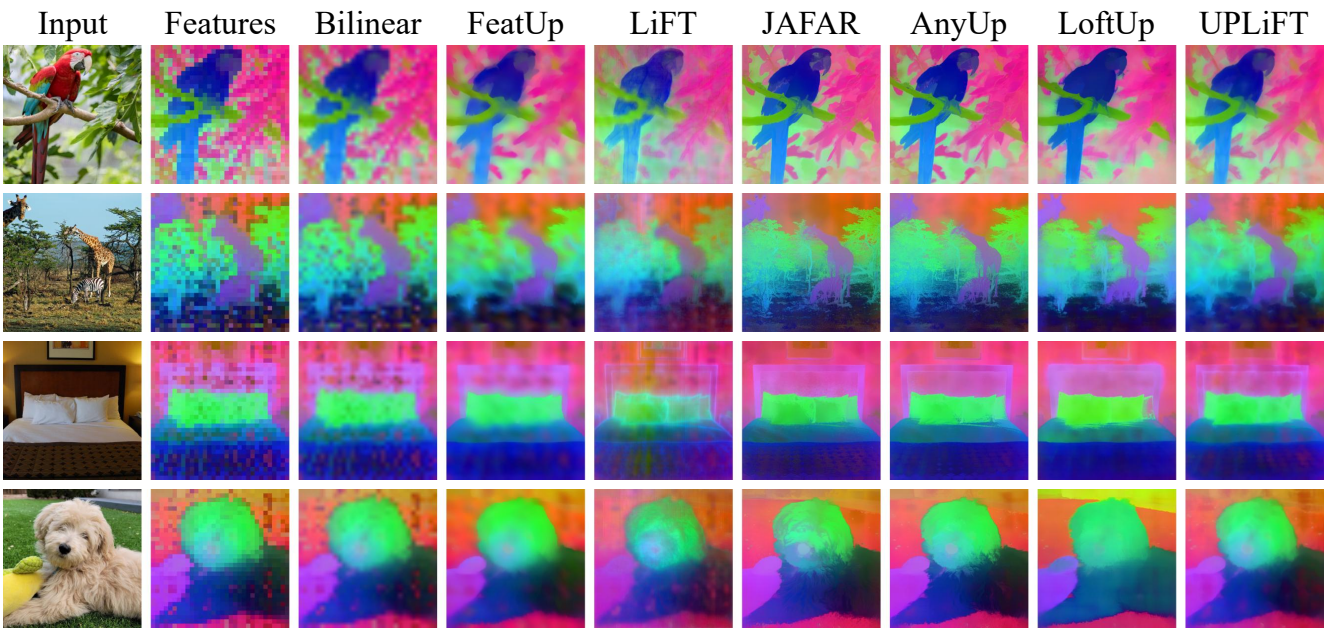

Figure 3. **Qualitative Feature Comparison.** We present PCA visualizations for various upsampling methods applied to DINOv2 features.

D.4. Additional Efficiency Comparisons

In Section 4, we assessed the efficiency of UPLiFT in comparison to recent state-of-the-art cross-attention-based feature upsamplers [2, 6, 18]. We present additional efficiency results here. Along with reporting inference time, we also report the inference GPU memory usage, and we also present these results for two different GPU types: an NVIDIA A5000 with 24 GB of memory, and an NVIDIA A6000 with 48 GB of memory. We plot these results in Figure 4. First, we see that the cross-attention-based methods clearly show quadratic scaling in their memory usage. All three methods reach the A5000's 24 GB memory limit at around 1500 visual tokens, while UPLiFT can reach over 2500 tokens. We note that at the lower range of image sizes, UPLiFT uses slightly more memory than the baseline methods, but by 1000 tokens, UPLiFT's linear scaling makes it more efficient. Moreover, the improved efficience of UPLiFT is even more apparent when tested on an A6000. With twice the GPU memory, UPLiFT can process twice as many visual tokens on an A6000, while the other methods rapidly fill the extra memory. The improved speed of UPLiFT for larger images is also apparent on the A6000, with it being $2-4\times$ faster than the baseline methods at 2000 tokens.

D.5. Semantic Stability with UPLiFT

A critical issue with LiFT [16] is the occurrence of semantic drift through iterative feature upsampling. We present a visual comparison of semantic drift in LiFT compared with the semantic stability achieved by UPLiFT. Following [4], we visualize the features of both methods using Principle Component Analysis (PCA) with 3 components. We extract the backbone features and all intermediate feature upsampling steps from both methods, and then perform a joint-PCA over the full collection. This means that consistent colors from one image to the next means consistent features too. We visualize the results in Figure 5.

Examining the PCA images for LiFT (row 1), we see that iterative upsampling leads to increasing degradation of the latents. The features become faded, murkier, and distorted through each step. Local image regions with consistent semantic content, like the dog's ear and nose region (row 3), do not maintain consistent and similar features. In comparison, UPLiFT (row 2) maintains consistent features through each upsampling step, without signs of internal distortions. In addition, the edges of objects are more distinctly defined than LiFT, and they match well to the original image, which enables better performance in tasks like semantic segmentation. Overall, these results demonstrate that our UPLiFT approach, through the use of Local Attender, is able to maintain consistent feature semantics through iterative upsampling steps to produce high-quality, pixel-dense features.

D.6. Additional Visualization for Generative Tasks

We provide further visualizations for UPLiFT applied to generative tasks. Figure 6 highlights UPLiFT's strong and efficient performance in $4\times$ super-resolution from 512×512 to 2048×2048. We find that UPLiFT adds only 8.47% latency compared to bilinear upsampling in latent space, and yields far superior image quality to the naive upsampling method. Figure 7 includes 2048×2048 images, upscaled from Stable Diffusion 1.5, and Figure 8 shows the

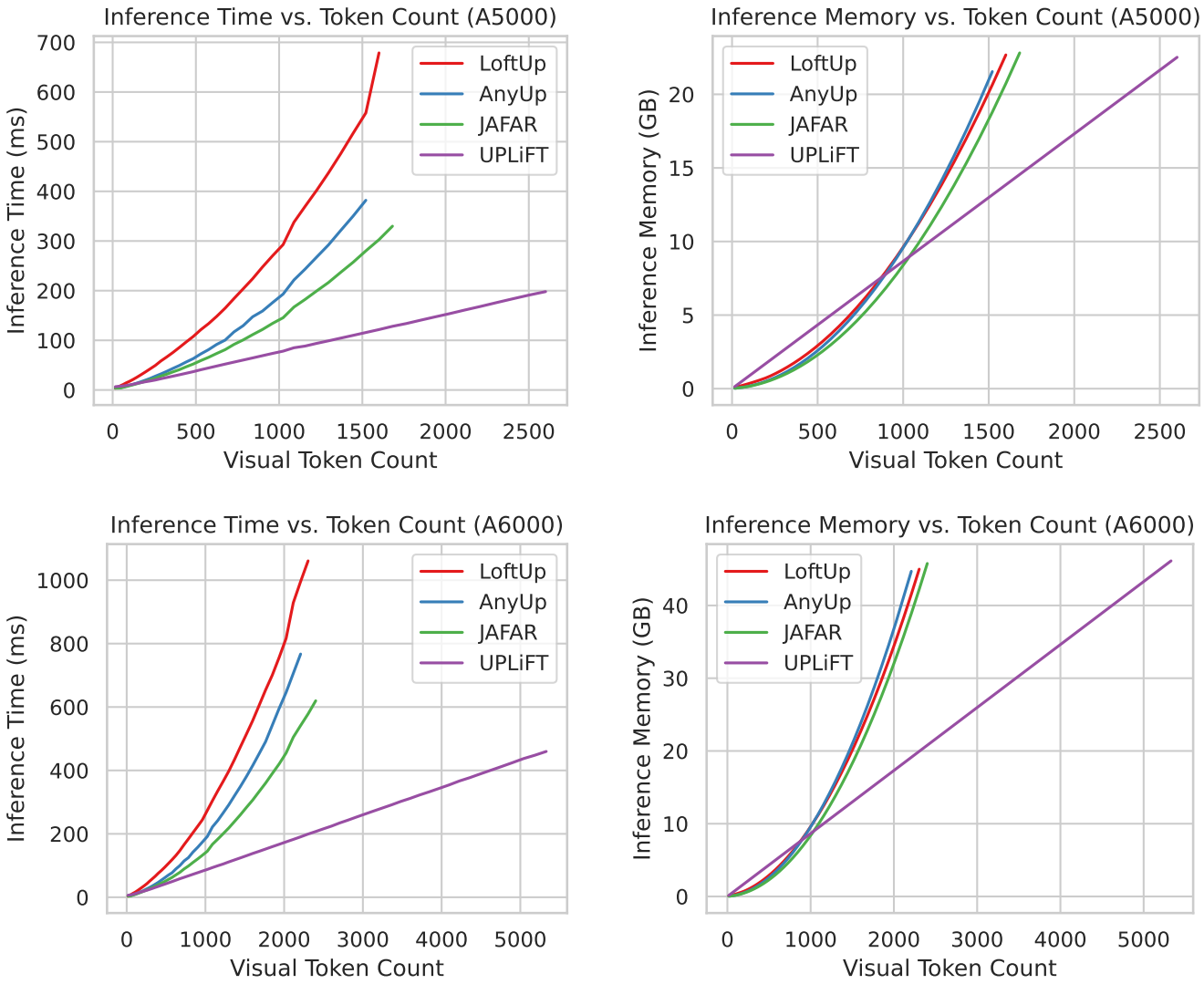

Figure 4. **Speed and Memory Usage Comparison.** We compare UPLiFT with recent cross-attention-based feature upsampling methods when running with gradually increasing image sizes. We report the average inference time and memory usage against the visual token count for two different GPU types: an NVIDIA A5000 with 24 GB of memory and an NVIDIA A6000 with 48 GB. UPLiFT maintains linear time and memory scaling with respect to the number of tokens, while the baseline methods show quadratic scaling. This gives UPLiFT far faster performance as the token count increases, and allows us to run pixel-dense feature upsampling on larger images.

same at 1024×1024. For the super-resolution task, we include samples from the evaluation datasets FacesHQ and LHQ using $2\times$ iterative upsampling twice, from 256×256 to 1024×1024 in Figure 9 and Figure 10 respectively.

References

[1] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016. 3

[2] Paul Couairon, Loick Chambon, Louis Serrano, Jean-Emmanuel Haugeard, Matthieu Cord, and Nicolas Thome. Jafar: Jack up any feature at any resolution. *arXiv preprint arXiv:2506.11136*, 2025. 1, 2, 7, 8

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009. 2

[4] Stephanie Fu, Mark Hamilton, Laura Brandt, Axel Feldman, Zhoutong Zhang, and William T Freeman. Featup: A model-agnostic framework for features at any resolution. *arXiv preprint arXiv:2403.10516*, 2024. 1, 7, 8, 10

[5] Ali Hassani, Steven Walton, Jiachen Li, Shen Li, and Humphrey Shi. Neighborhood attention transformer. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2023. 6

[6] Haiwen Huang, Anpei Chen, Volodymyr Havrylov, Andreas Geiger, and Dan Zhang. Loftup: Learning a coordinate-

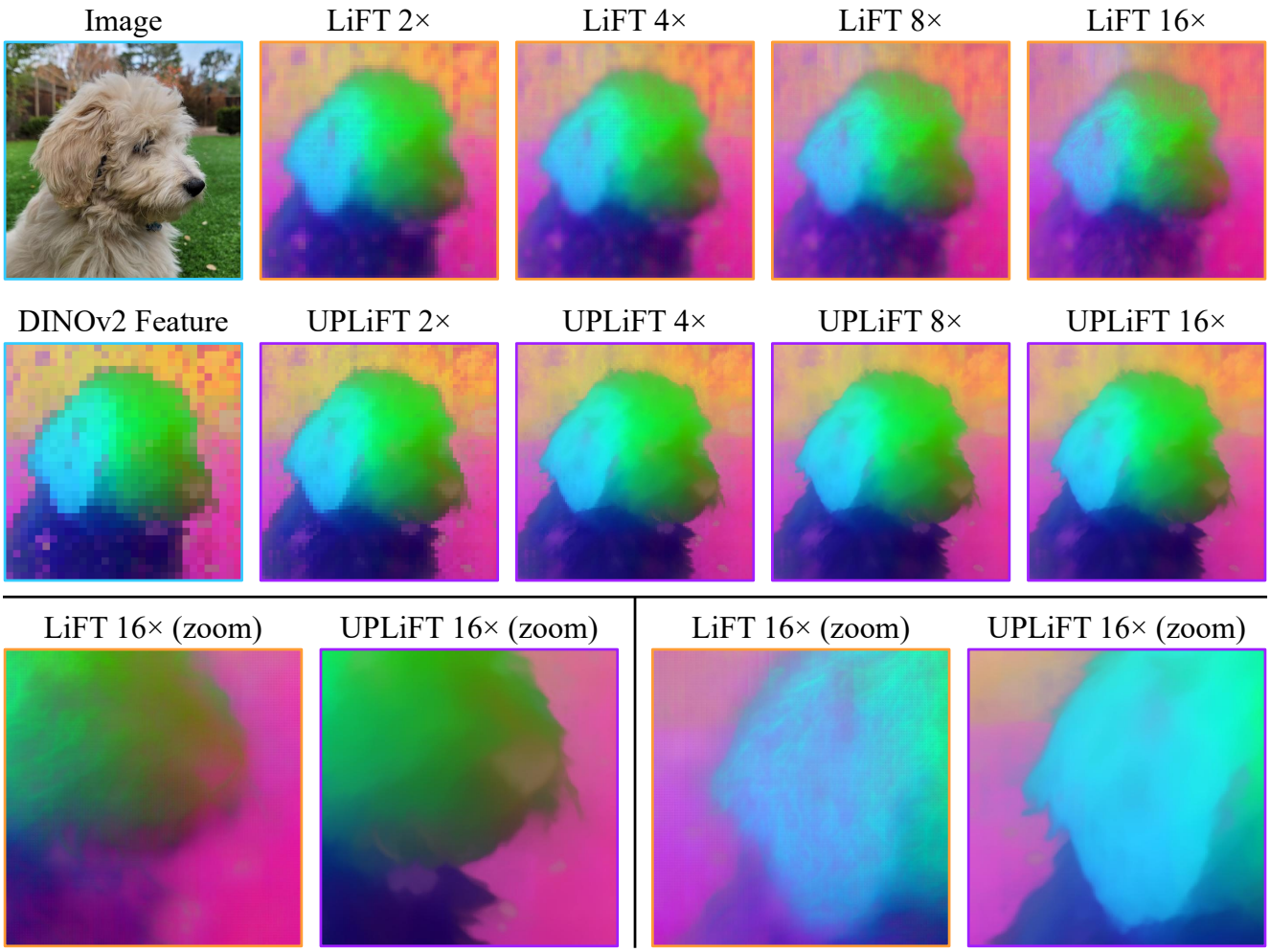

Figure 5. **Comparison of semantic drift in LiFT and semantic stability in UPLiFT.** We visualize intermediate feature upsampling steps through PCA, following [4]. LiFT shows signs of feature drift, with local features becoming murkier and more distorted in deeper steps. This drift can lead to poor performance in downstream tasks, as the strength of the original backbone representation is lost. UPLiFT maintains consistent feature semantics thanks to our Local Attender, and local object regions maintain consistent features (coloration) across all upsampling stages.

based feature upsampler for vision foundation models. *arXiv preprint arXiv:2504.14032*, 2025. 1, 2, 7, 8

[7] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. pmlr, 2015. 3

[8] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4015–4026, 2023. 2, 7

[9] Johannes Kopf, Michael F Cohen, Dani Lischinski, and Matt Uyttendaele. Joint bilateral upsampling. *ACM Transactions on Graphics (ToG)*, 26(3):96–es, 2007. 1

[10] Simian Luo, Yiqin Tan, Suraj Patil, Daniel Gu, Patrick Von Platen, Apolinário Passos, Longbo Huang, Jian Li, and Hang Zhao. Lcm-lora: A universal stable-diffusion acceleration module. *arXiv preprint arXiv:2311.05556*, 2023. 4

[11] Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, et al. Dinov2: Learning robust visual features without supervision. *arXiv preprint arXiv:2304.07193*, 2023. 1, 2, 7

[12] Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis. *arXiv preprint arXiv:2307.01952*, 2023. 4

[13] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2021. 4

[14] Johannes Schusterbauer, Ming Gui, Pingchuan Ma, Nick Stracke, Stefan Andreas Baumann, Vincent Tao Hu, and Björn Ommer. Fmboost: Boosting latent diffusion with flow matching. In *European Conference on Computer Vision*, pages 338–355. Springer, 2024. 3, 4, 5, 16

[15] Oriane Siméoni, Huy V Vo, Maximilian Seitzer, Federico Baldassarre, Maxime Oquab, Cijo Jose, Vasil Khalidov, Marc Szafraniec, Seungeun Yi, Michaël Ramamonjisoa, et al. Dinov3. *arXiv preprint arXiv:2508.10104*, 2025. 2, 7

[16] Saksham Suri, Matthew Walmer, Kamal Gupta, and Abhinav Shrivastava. Lift: A surprisingly simple lightweight feature transform for dense vit descriptors. In *European Conference on Computer Vision*, pages 110–128. Springer, 2024. 1, 8

[17] 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*, 13(4):600–612, 2004. 4

[18] Thomas Wimmer, Prune Truong, Marie-Julie Rakotosaona, Michael Oechsle, Federico Tombari, Bernt Schiele, and Jan Eric Lenssen. Anyup: Universal feature upsampling. *arXiv preprint arXiv:2510.12764*, 2025. 1, 2, 7, 8

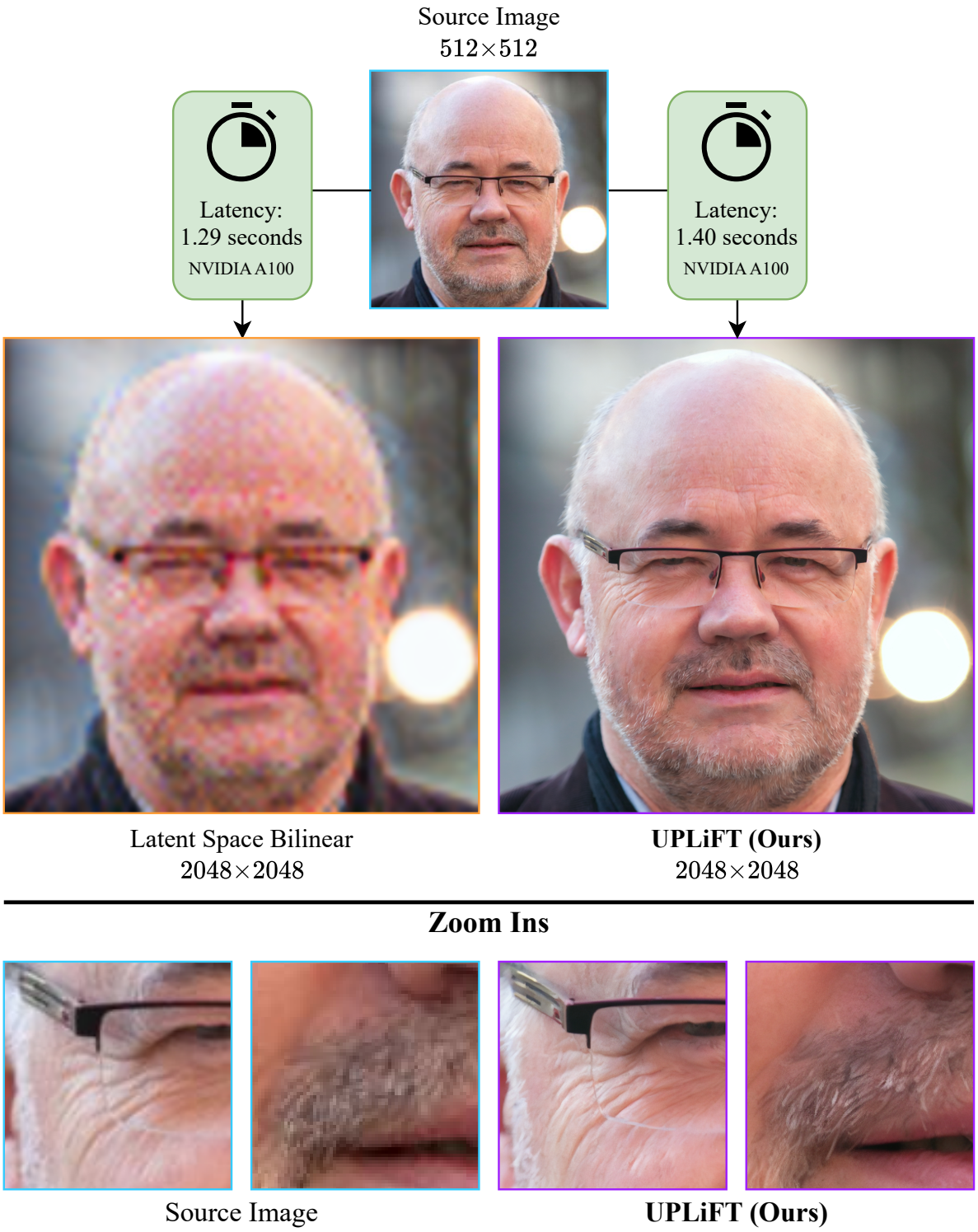

Figure 6. **UPLiFT *vs*. Latent Space Bilinear** $4\times$ **upsampling for image super-resolution.** We compare the latency of UPLiFT versus applying bilinear upsampling in latent space for image super-resolution to demonstrate UPLiFT's state-of-the-art efficiency. While only 8.47% slower end-to-end, UPLiFT produces significantly better visual fidelity, as shown by the zoomed-in views (bottom) which display the source low-resolution image compared with UPLiFT's super-resolution image. The low-resolution image is selected from the FacesHQ dataset and bilinearly downscaled from 1024×1024 to 512×512 before UPLiFT is applied. Best viewed zoomed in.

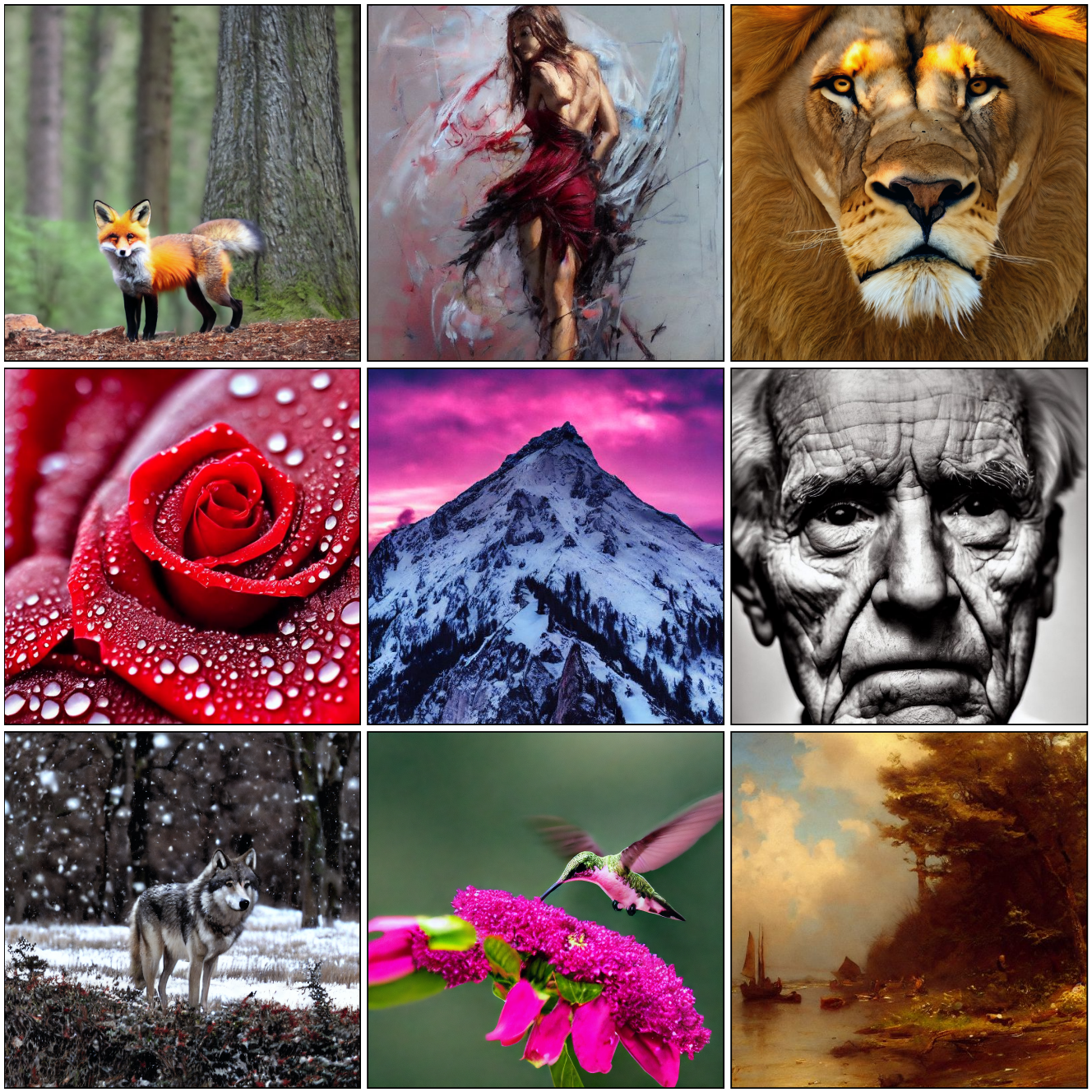

Figure 7. **UPLiFT $512\times512 \rightarrow 2048\times2048$ upsampled images using Stable Diffusion 1.5.** We apply our VAE UPLiFT model to this task in a $4\times$ upsampling configuration. UPLiFT upsamples latents corresponding to 512×512 images generated using 50 diffusion steps on Stable Diffusion 1.5. Best viewed zoomed in.

Figure 8. **UPLiFT $512\times512 \rightarrow 1024\times1024$ upsampled images using Stable Diffusion 1.5.** We apply our VAE UPLiFT model to this task in a $2\times$ upsampling configuration. UPLiFT upsamples latents corresponding to 512×512 images generated using 50 diffusion steps on Stable Diffusion 1.5, and the end-to-end latency is 2.75 seconds on an NVIDIA A100 GPU. The UPLiFT model itself takes only 104 milliseconds of this time. Best viewed zoomed in.

Figure 9. **UPLiFT $256\times256 \rightarrow 1024\times1024$ super-resolution samples from FacesHQ.** We use our VAE UPLiFT model that is *not fine-tuned* for image super-resolution and is only trained in latent space. Our end-to-end upsampling time is only **270.9 milliseconds** on an NVIDIA A100 GPU. Best viewed zoomed in.

Figure 10. **UPLiFT $256\times256 \rightarrow 1024\times1024$ super-resolution samples from LHQ.** We use our VAE UPLiFT model, which is trained as a generalist model and is not specifically fine-tuned for this dataset. The LHQ dataset presents a greater challenge than FacesHQ, based on the diversity of visual textures present. Despite this challenge, we see good performance with our generalist UPLiFT. In comparison, [14] uses a fine-tuned model with $3\times$ the parameter count for evaluations on LHQ versus FacesHQ. Best viewed zoomed in.