Dressing in the Wild by Watching Dance Videos Supplementary Material

Xin Dong[1], Fuwei Zhao[2], Zhenyu Xie[2], Xijin Zhang[1]
Daniel K. Du[1], Min Zheng[1], Xiang Long[1], Xiaodan Liang[2*], Jianchao Yang[1]
[1]ByteDance, [2]Shenzhen Campus of Sun Yat-Sen University
{zhaofw@mail2,xiezhy6@mail2,xdliang328@mail}.sysu.edu.cn
{dongxin.1016,zhangxijin,dukang.daniel,zhengmin.666,longxiang.0,yangjianchao}@bytedance.com

1. Architecture Details

1.1. Stage 1: Conditional Person Segmentation

This stage is composed of two separate encoders and one upsampling decoder bridged by a serious of residual blocks. We provide in Table. 1 the detailed architecture of the three components.

1.2. Stage 2: Pixel Flow Estimation

In this stage, we adopt the same FlowNetCorr architecture proposed in [3] as our Pixel Flow Network (PFN). We give its detailed overview in Table. 2 but refer readers to the original paper for more motivation and details especially on the feature correlation layer.

1.3. Stage 3: Garment Transfer with Blend Flow

We provide network details of this stage containing the background inpainter G^B, the reconstruction generator G^S and the try-on generator G^T in Table. 3. Note, the architecture details of G^S and G^T are identical, except the dimension of the inputs (i.e., 82 for G^S and 85 for G^T).

2. Dataset Details

Collection and clean process. We first search for enormous candidate dance videos on the public internet, and then run on them the human detector [1] to filter out multi-person results. Thereafter, we train Mask-RCNN [5] on the DeepFashion2 dataset [4] to detect and classify the clothes attributes, which is used to balance the distribution of garment types among the videos. Finally, we construct a large-scale dataset named *Dance50k* of $50,000$ single-person dance sequences (about 15s duration) that features diverse poses and rich garment types.

Training/Testing Split. Since we aim at image-based garment transfer, the video frames need to be sampled forming the training/testing image collections. After applying the sampling and asserting process described in the main paper (Section.4.2), we get a split of 949626 and 15815 images respectively for training and testing.

Data Examples. Fig. 1 shows examples from the *Dance50k*, at which we can see most videos are taken in the wild with clear person foreground.

3. User Study Details

We conduct a user evaluation study to assess the quality of the garment transfer results. Specifically, 40 volunteers are invited to complete a questionnaire that contains 30 assignments. In each assignment, given a source person image and a query person image, the volunteers are required to select the most realistic garment transfer image out of 2 choices, which are synthesized by our method and the LWG [6] respectively. We do not evaluate the results of ADGAN [7] and DiOR [2] for the user study because they have no background. Please refer to Sec.4.3 in the main text for the detailed quantitative results of the user study.

4. Additional Visual Results

We provide more qualitative comparisons of the garment transfer results in Fig. 2, and single-item try-on result in Fig. 3. Note, for half-to-full-body garment transfer, we believe this would be a meaningful but ill-posed problem. Transferring a cropped garment (e.g., a half pant) to a full-body person heavily relies on imagination of the missing source texture. While our method has the potential to achieve that imagination, ensuring num (I_s) >num (I_t) (as described in the main text) is more practical valuable as people always want the desired clothes is intact.

In Fig. 4, we further show the result comparison between ours and the related work LWG [6] on the iPER dataset proposed in [6]. The iPER dataset is relatively simple (especially the background) and LWG has somewhat overfitted on this small dataset ($\sim$30 videos). Our new *Dance50k* dataset ($\sim$50k videos) is much more diverse than iPER, and

the garment transfer results of LWG trained on *Dance50k* is inferior to our wFlow (see Fig.4 in the main paper), showing its powerlessness when scaling to large dataset with loose garments and cluttered background.

Though the original purpose is image-based garment transfer, our method can also generalize to video virtual try-on in a frame-by-frame generation manner. We provide additional video results at https://figshare.com/s/9ceebc27955cb82a3954.

References

[1] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. IEEE Transactions on Pattern Analysis and Machine Intelligence, 2019. 1

[2] Aiyu Cui, Daniel McKee, and Svetlana Lazebnik. Dressing in order: Recurrent person image generation for pose transfer, virtual try-on and outfit editing. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), pages 14638–14647, October 2021. 1

[3] A. Dosovitskiy, P. Fischer, Eddy Ilg, Philip Häusser, Caner Hazirbas, V. Golkov, P. V. D. Smagt, D. Cremers, and T. Brox. Flownet: Learning optical flow with convolutional networks. 2015 IEEE International Conference on Computer Vision (ICCV), pages 2758–2766, 2015. 1

[4] Yuying Ge, Ruimao Zhang, Lingyun Wu, Xiaogang Wang, Xiaoou Tang, and Ping Luo. A versatile benchmark for detection, pose estimation, segmentation and re-identification of clothing images. CVPR, 2019. 1

[5] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross B. Girshick. Mask r-cnn. IEEE Transactions on Pattern Analysis and Machine Intelligence, 42:386–397, 2020. 1

[6] Wen Liu, Zhixin Piao, Jie Min, Wenhan Luo, Lin Ma, and Shenghua Gao. Liquid warping gan: A unified framework for human motion imitation, appearance transfer and novel view synthesis. In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), October 2019. 1, 7, 9

[7] Yifang Men, Yiming Mao, Yuning Jiang, Wei-Ying Ma, and Zhouhui Lian. Controllable person image synthesis with attribute-decomposed gan. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2020. 1, 7

CSN			
Layer		**Type**	**Output Size**
Source Encoder	Input 1	Input	(512,512,83)
	Conv 1-1	Conv2d 7×7, InstanceNorm, ReLU	(512,512,64)
	Conv 1-2	Conv2d 3×3, InstanceNorm, ReLU	(256,256,128)
	Conv 1-3	Conv2d 3×3, InstanceNorm, ReLU	(128,128,256)
	Conv 1-4	Conv2d 3×3, InstanceNorm, ReLU	(64,64,512)
Target Encoder	Input 2	Input	(512,512,79)
	Conv 2-1	Conv2d 7×7, InstanceNorm, ReLU	(512,512,64)
	Conv 2-2	Conv2d 3×3, InstanceNorm, ReLU	(256,256,128)
	Conv 2-3	Conv2d 3×3, InstanceNorm, ReLU	(128,128,256)
	Conv 2-4	Conv2d 3×3, InstanceNorm, ReLU	(64,64,512)
Residual Blocks	Concate	Concate (Conv 1-4, Conv 2-4)	(64,64,1024)
	Res 1	Residual Block with Conv2d 3×3, InstanceNorm, ReLU	(64,64,1024)
	Res 2	Residual Block with Conv2d 3×3, InstanceNorm, ReLU	(64,64,1024)
	Res 3	Residual Block with Conv2d 3×3, InstanceNorm, ReLU	(64,64,1024)
	Res 4	Residual Block with Conv2d 3×3, InstanceNorm, ReLU	(64,64,1024)
	Res 5	Residual Block with Conv2d 3×3, InstanceNorm, ReLU	(64,64,1024)
	Res 6	Residual Block with Conv2d 3×3, InstanceNorm, ReLU	(64,64,1024)
Decoder	Upsample 1	ConvTranspose2d 3×3, InstanceNorm, ReLU	(128,128,512)
	Skip Connection 1	Skip Connection (Upsample 1, Conv 1-3, Conv 2-3)	(128,128,1024)
	Conv 3-1	Conv2d 1×1, InstanceNorm, ReLU	(128,128,512)
	Upsample 2	ConvTranspose2d 3×3, InstanceNorm, ReLU	(256,256,256)
	Skip Connection 2	Skip Connection (Upsample 2, Conv 1-2, Conv 2-2)	(256,256,512)
	Conv 3-2	Conv2d 1×1, InstanceNorm, ReLU	(256,256,256)
	Upsample 3	ConvTranspose2d 3×3, InstanceNorm, ReLU	(512,512,128)
	Skip Connection 3	Skip Connection (Upsample 3, Conv 1-1, Conv 2-1)	(512,512,256)
	Conv 3-3	Conv2d 1×1, InstanceNorm, ReLU	(512,512,128)
	Conv 3-4	Conv2d 7×7, input from Conv 3-3	(512,512,2)
	Conv 3-5	Conv2d 7×7, input from Conv 3-3	(512,512,20)

Table 1. The architecture details of Conditional Segmentation Network (CSN).

PFN			
Layer		**Type**	**Output Size**
Source Encoder	Input 1	Input	(512,512,83)
	Conv 1-1	Conv2d 7×7, InstanceNorm, LeakyReLU	(256,256,64)
	Conv 1-2	Conv2d 5×5, InstanceNorm, LeakyReLU	(128,128,128)
	Conv 1-3	Conv2d 5×5, InstanceNorm, LeakyReLU	(64,64,256)
	Conv redir	Conv2d 1×1, InstanceNorm, LeakyReLU	(64,64,32)
	Correlation	Correlation between (Conv 1-3, Conv 2-3)	(64,64,441)
	Concate	Concate (Conv redir, Correlation)	(64,64,473)
	Conv 1-3-1	conv2d 3×3, InstanceNorm, LeakyReLU	(64,64,256)
	Conv 1-4	conv2d 3×3, InstanceNorm, LeakyReLU	(32,32,512)
	Conv 1-4-1	conv2d 3×3, InstanceNorm, LeakyReLU	(32,32,512)
	Conv 1-5	conv2d 3×3, InstanceNorm, LeakyReLU	(16,16,512)
	Conv 1-5-1	conv2d 3×3, InstanceNorm, LeakyReLU	(16,16,512)
	Conv 1-6	conv2d 3×3, InstanceNorm, LeakyReLU	(8,8,1024)
	Conv 1-6-1	conv2d 3×3, InstanceNorm, LeakyReLU	(8,8,1024)
Target Encoder	Input 2	Input	(512,512,83)
	Conv 2-1	Conv2d 7×7, InstanceNorm, LeakyReLU	(256,256,64)
	Conv 2-2	Conv2d 5×5, InstanceNorm, LeakyReLU	(128,128,128)
	Conv 2-3	Conv2d 5×5, InstanceNorm, LeakyReLU	(64,64,256)
Flow Estimation Module	Flow 6	Conv2d 3×3, input from Conv 1-6-1	(8,8,2)
	Upsample 5	ConvTranspose2d 4×4, IN, LeakyReLU, input from Conv 1-6-1	(16,16,512)
	Upsample 5-1	ConvTranspose2d 4×4, input from Flow 6	(16,16,2)
	Concate 5	Concate (Conv 1-5-1, Upsample 5, Upsample 5-1)	(16,16,1026)
	Flow 5	Conv2d 3×3, inputs from Concate 5	(16,16,2)
	Upsample 4	ConvTranspose2d 4×4, IN, LeakyReLU, input from Concate 5	(32,32,256)
	Upsample 4-1	ConvTranspose2d 4×4, input from Flow 5.	(32,32,2)
	Concate 4	Concate (Conv 1-4-1, Upsample 4, Upsample 4-1)	(32,32,770)
	Flow 4	Conv2d 3×3, inputs from Concate 4	(32,32,2)
	Upsample 3	ConvTranspose2d 4×4, IN, LeakyReLU, input from Concate 4	(64,64,128)
	Upsample 3-1	ConvTranspose2d 4×4, input from Flow 4	(64,64,2)
	Concate 3	Concate (Conv 1-3-1, Upsample 3, Upsample 3-1)	(64,64,386)
	Flow 3	Conv2d 3×3, inputs from Concate 3	(64,64,2)
	Upsample 2	ConvTranspose2d 4×4, IN, LeakyReLU, input from Concate 3	(128,128,64)
	Upsample 2-1	ConvTranspose2d 4×4, input from Flow 3	(128,128,2)
	Concate 2	Concate (Conv 1-2, Upsample 2, Upsample 2-1)	(128,128,194)
	Flow 2	Conv2d 3×3, inputs from Concate 2	(128,128,2)
	Upsample 1	ConvTranspose2d 4×4, IN, LeakyReLU, input from Concate 2	(256,256,32)
	Upsample 1-1	ConvTranspose2d 4×4, input from Flow 2	(256,256,2)
	Concate 1	Concate (Conv 1-1, Upsample 1, Upsample 1-1)	(256,256,98)
	Flow 1	Conv2d 3×3, inputs from Concate 2	(256,256,2)

Table 2. The architecture details of Pixel Flow Network (PFN).

G^S (G^T)			
Layer		**Type**	**Output Size**
Encoder	Input	Input	(512,512,82(85))
	Conv 1-1	Conv2d 7×7, InstanceNorm, ReLU	(512,512,64)
	Conv 1-2	Conv2d 3×3, InstanceNorm, ReLU	(256,256,128)
	Conv 1-3	Conv2d 3×3, InstanceNorm, ReLU	(128,128,256)
	Conv 1-4	Conv2d 3×3, InstanceNorm, ReLU	(64,64,512)
Residual Blocks	Res 1-1	Residual Block with Conv2d 3×3, InstanceNorm, ReLU	(64,64,512)
	Res 1-2	Residual Block with Conv2d 3×3, InstanceNorm, ReLU	(64,64,512)
	Res 1-3	Residual Block with Conv2d 3×3, InstanceNorm, ReLU	(64,64,512)
	Res 1-4	Residual Block with Conv2d 3×3, InstanceNorm, ReLU	(64,64,512)
	Res 1-5	Residual Block with Conv2d 3×3, InstanceNorm, ReLU	(64,64,512)
	Res 1-6	Residual Block with Conv2d 3×3, InstanceNorm, ReLU	(64,64,512)
Decoder	UpSample 1-1	ConvTranspose2d 3×3, InstanceNorm, ReLU	(128,128,256)
	Skip Connection 1	Skip Connection (Upsample 1, Conv 1-3)	(128,128,512)
	Conv 2-1	Conv2d 1×1, InstanceNorm, ReLU	(128,128,256)
	UpSample 1-2	ConvTranspose2d 3×3, InstanceNorm, ReLU	(256,256,128)
	Skip Connection 2	Skip Connection (Upsample 2, Conv 1-2)	(256,256,256)
	Conv 2-2	Conv2d 1×1, InstanceNorm, ReLU	(256,256,128)
	UpSample 1-3	ConvTranspose2d 3×3, InstanceNorm, ReLU	(512,512,64)
	Skip Connection 3	Skip Connection (Upsample 3, Conv 1-1)	(512,512,128)
	Conv 2-3	Conv2d 1×1, InstanceNorm, ReLU	(512,512,64)
	Conv 2-4	Conv2d 7×7, Tanh, input from Conv 2-3	(512,512,3)
	Conv 2-5	Conv2d 7×7, Sigmoid, input from Conv 2-3	(512,512,1)
	Conv 2-6	Conv2d 7×7, Sigmoid, input from Conv 2-3	(512,512,1)
	Conv 2-7	Conv2d 7×7, Sigmoid, input from Conv 2-3	(512,512,1)
G^B			
Layer		**Type**	**Output Size**
Encoder	Input	Input	(512,512,4)
	Conv 3-1	Conv2d 7×7, InstanceNorm, ReLU	(512,512,64)
	Conv 3-2	Conv2d 3×3, InstanceNorm, ReLU	(256,256,128)
	Conv 3-3	Conv2d 3×3, InstanceNorm, ReLU	(128,128,256)
	Conv 3-4	Conv2d 3×3, InstanceNorm, ReLU	(64,64,512)
Residual Blocks	Res 2-1	Residual Block with Conv2d 3×3, InstanceNorm, ReLU	(64,64,512)
	Res 2-2	Residual Block with Conv2d 3×3, InstanceNorm, ReLU	(64,64,512)
	Res 2-3	Residual Block with Conv2d 3×3, InstanceNorm, ReLU	(64,64,512)
	Res 2-4	Residual Block with Conv2d 3×3, InstanceNorm, ReLU	(64,64,512)
	Res 2-5	Residual Block with Conv2d 3×3, InstanceNorm, ReLU	(64,64,512)
	Res 2-6	Residual Block with Conv2d 3×3, InstanceNorm, ReLU	(64,64,512)
Decoder	UpSample 2-1	ConvTranspose2d 3×3, InstanceNorm, ReLU	(128,128,256)
	UpSample 2-2	ConvTranspose2d 3×3, InstanceNorm, ReLU	(256,256,128)
	UpSample 2-3	ConvTranspose2d 3×3, InstanceNorm, ReLU	(512,512,64)
	Conv 4-1	Conv2d 7×7, Tanh	(512,512,3)

Table 3. The architecture details of reconstruction generator G^S, try-on generator G^T, and background inpainter G^B.

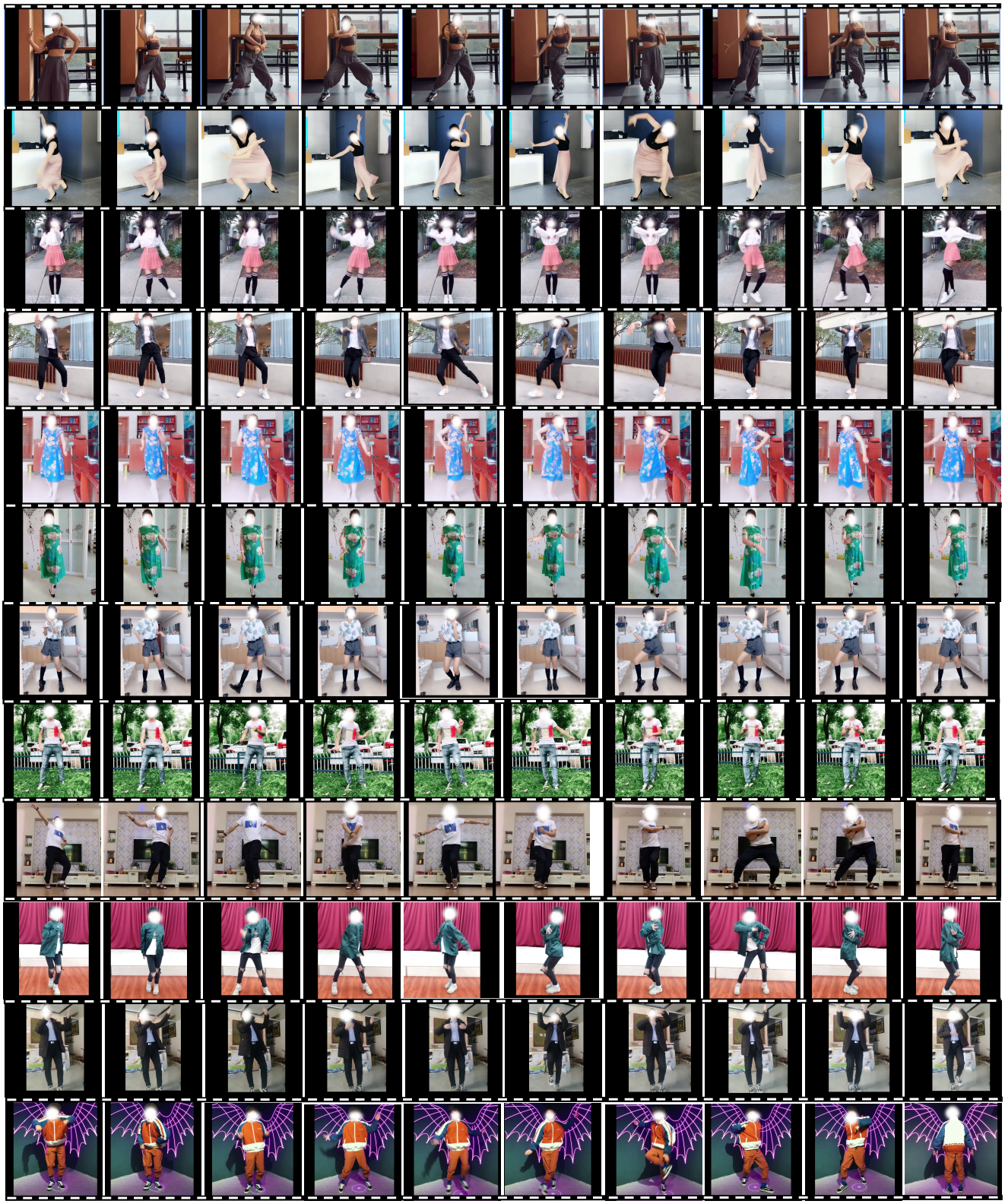

Figure 1. Examples of our collected Dance50k dataset covering diverse dance poses and a wide variety of garments.

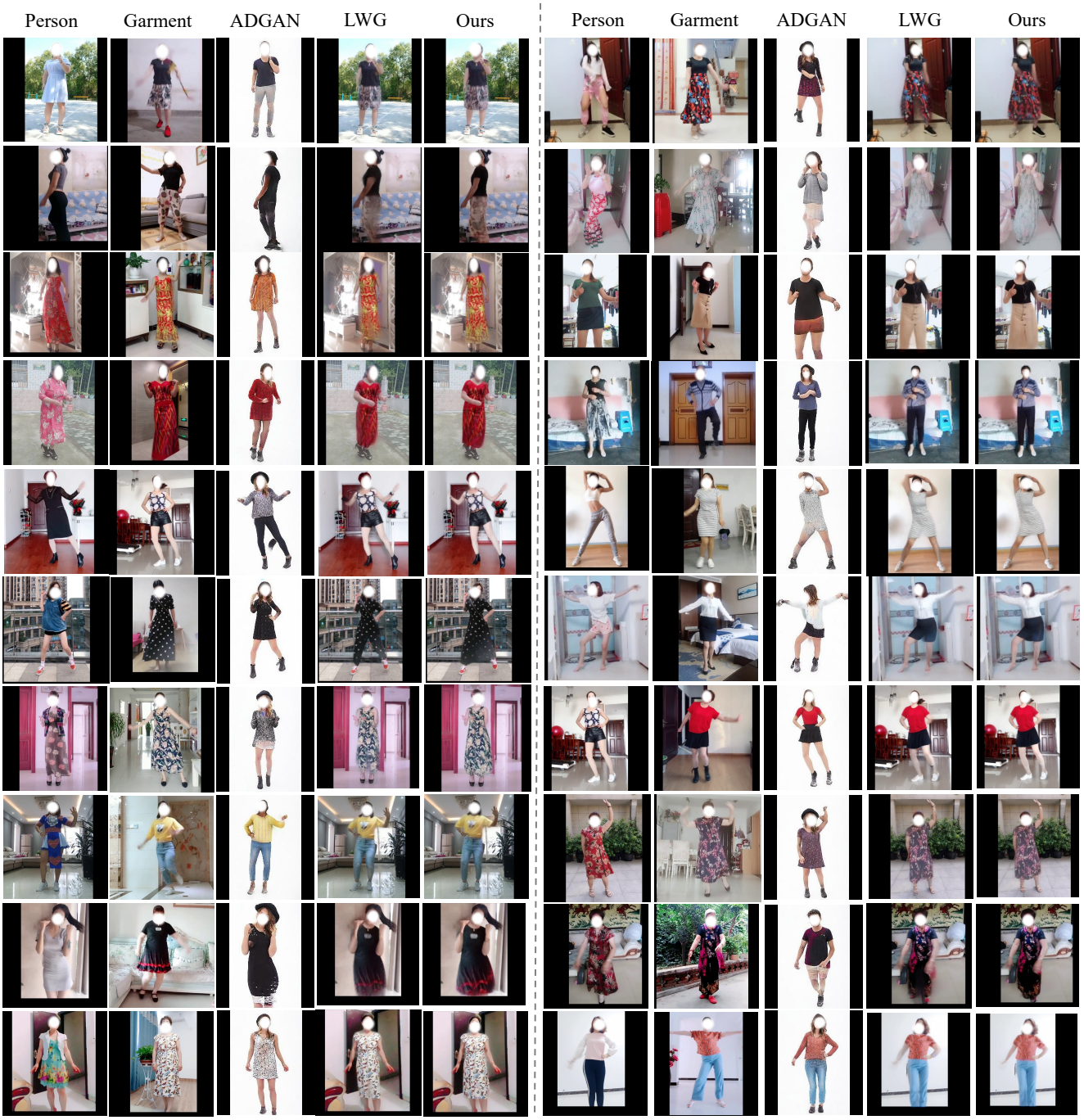

Figure 2. Qualitative comparisons on *Dance50k*. The first two columns represent the inputs, while the others are garment transfer results from our method and the other two baselines (LWG [6] and ADGAN [7]. Ours contain richer texture details and more successfully transfer the loose garments.

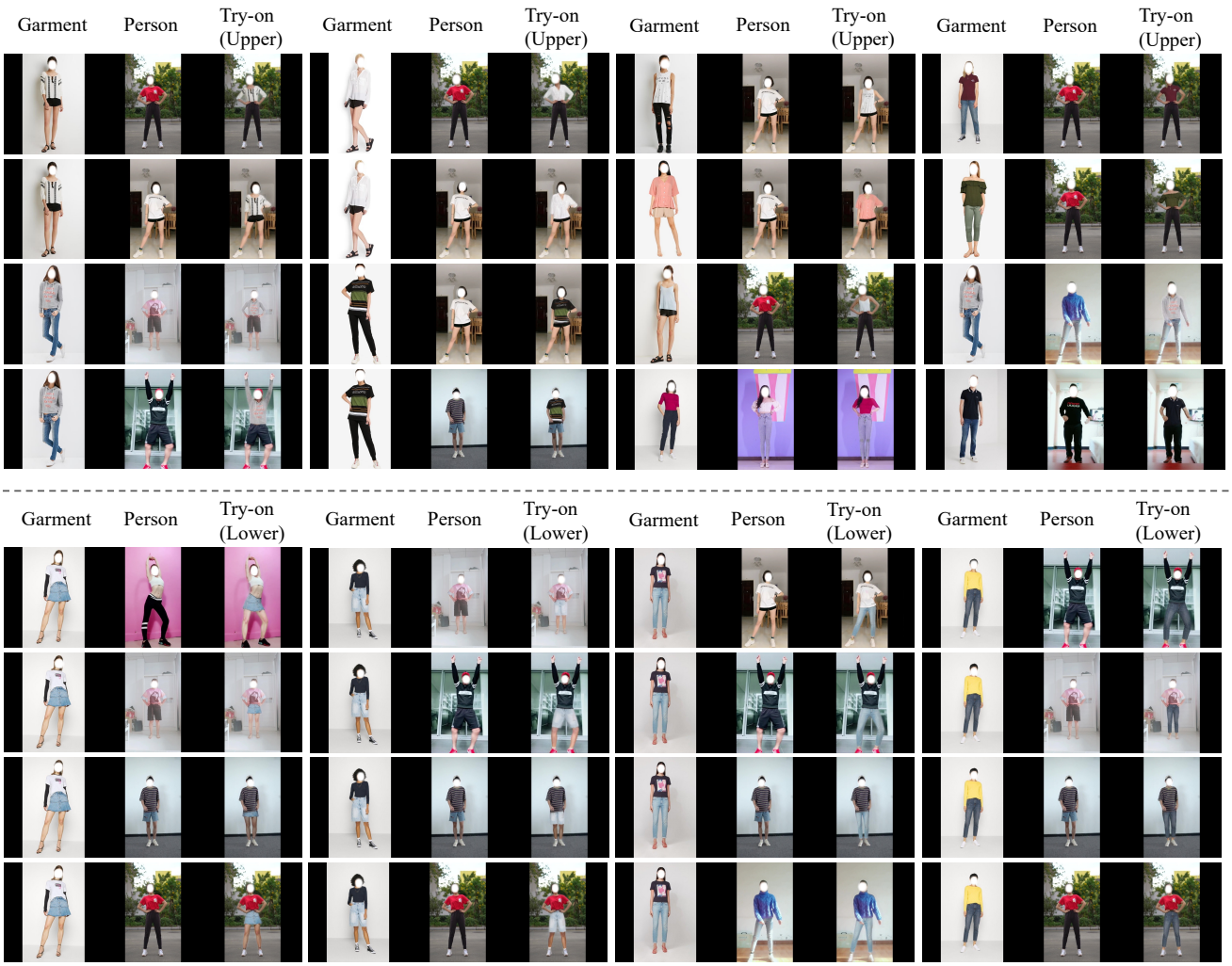

Figure 3. By setting appropriate protected body part, our model also supports single-item transfer.

Figure 4. Visual Comparison on the iPER Dataset between wFlow and LWG [6]