ARShoe: Real-Time Augmented Reality Shoe Try-on System on Smartphones

Shan An1,2, Guangfu Che1, Jinghao Guo1, Haogang Zhu2,3†, Junjie Ye1, Fangru Zhou1, Zhaoqi Zhu1, Dong Wei1, Aishan Liu2, Wei Zhang1
1JD.COM Inc., Beijing, China
2State Key Lab of Software Development Environment, Beihang University, Beijing, China
3Beijing Advanced Innovation Center for Big Data-Based Precision Medicine, Beijing, China

{anshan, cheguangfu1, guojinghao1, yejunjie12, zhufangru, zhuzhaoqi, weidong53, zhangwei96}@jd.com
{haogangzhu, liuaishan}@buaa.edu.cn

ABSTRACT
Virtual try-on technology enables users to try various fashion items using augmented reality and provides a convenient online shopping experience. However, most previous works focus on the virtual try-on for clothes while neglecting that for shoes, which is also a promising task. To this concern, this work proposes a real-time augmented reality virtual shoe try-on system for smartphones, namely ARShoe. Specifically, ARShoe adopts a novel multi-branch network to realize pose estimation and segmentation simultaneously. A solution to generate realistic 3D shoe model occlusion during the try-on process is presented. To achieve a smooth and stable try-on effect, this work further develop a novel stabilization method. Moreover, for training and evaluation, we construct the very first large-scale foot benchmark with multiple virtual shoe try-on task-related labels annotated. Exhaustive experiments on our newly constructed benchmark demonstrate the satisfying performance of ARShoe. Practical tests on common smartphones validate the real-time performance and stabilization of the proposed approach.

CCS CONCEPTS
• Human-centered computing → Mixed / augmented reality;
• Computing methodologies → Image-based rendering; Image segmentation; Neural networks.

KEYWORDS
Augmented reality; Virtual try-on; Pose estimation; Segmentation; Stabilization; Benchmark

1 INTRODUCTION
Recent years have witnessed the huge commercial potential of online shopping. Considerable research efforts have been put towards online shopping-related technologies, e.g., clothing retrieval [17], product recommendation [34], and virtual try-on [31]. Dedicating to facilitate the online shopping experience of fashion items, virtual try-on is a promising technology, yet still challenging.

Previous works mainly focus on the virtual try-on for clothes [12, 15, 30, 31, 40]. In [31], a fully convolutional graph neural network is proposed to achieve a more realistic virtual try-on effect with fine-grained details. Chia-Wei Hsieh et al. [15] propose a novel FashionOn network to tackle the body occlusion problem while

Corresponding author.

Figure 1: (a) Virtual shoe try-on results generated by ARShoe on a smartphone. The last row shows the target shoes, while the first column shows the original input. (b) The speed of our virtual shoe try-on system on common smartphones. ARShoe realizes a realistic virtual try-on effect with a real-time speed of over 30 frames/s (FPS) on smartphones.
preserving the detailed information of clothes. Utilizing a coarse-to-fine strategy, VITON [12] learns to synthesize a photo-realistic image of target clothes overlaid on a person in a specific pose. Comparing to clothes virtual try-on, virtual try-on for shoes has not been studied much so far while it is also a pivotal task. Chao-Te Chou et al. [7] focus on synthesizing images with shoes in the source photo replaced by target fashion shoes.

Since the main users of online shopping are consumers with smartphones, therefore smartphones are supposed to be a common platform of a virtual try-on system, which raises stringent requirements of real-time performance and computational efficiency of adopted algorithms. However, existing approaches commonly put great effort to optimize the realistic performance with severe computational burn introduced, while neglecting that real-time performance is also a crucial metric that influences user satisfaction. In addition to limited computation resources, the sensing device on smartphones that can be used for virtual try-on is usually only a monocular camera. In that case, the adopted algorithm is also expected to realize 6-DoF pose estimation accurately and efficiently. Moreover, considering the usage scenarios and the user’s shooting angle, a system of virtual shoe try-on for common smartphones is more promising and needs comparing to virtual clothes try-on.

To this end, this work dedicates to study an efficient virtual shoe try-on system. In our concern, the task of virtual shoe try-on involves the following crucial subtasks: 1) 6-DoF pose estimation of feet wearing shoes. To generate 3D shoe models with the correct viewpoint, accurate 6-DoF feet pose estimation is important. Commonly, pose from 2D images is estimated by mapping 2D keypoints through the Perspective-n-Point (PnP) algorithm. 2) 2D pose estimation. Different from virtual clothes try-on, there are always two instances in the usage scenarios of virtual shoe try-on. Therefore, the 2D poses of feet are estimated to guide the accurate grouping of 2D keypoints into each foot. 3) Occlusion generation. To produce a realistic virtual render effect, the overlays of target shoes should appear in a precise position with an appropriate scale. 4) Stabilization. Since we target real-time virtual try-on, frequent jitter of the overlaid shoes on the screen will decrease the user experience.

In this work, we construct a neat and novel framework namely ARShoe to realize the above subtasks simultaneously and efficiently. Specifically, an efficient and effective multi-branch network is proposed to learn multi-tasks jointly, i.e., 2D keypoints detection, 2D pose estimation, and segmentation. To achieve realistic occlusion, we utilize the segmentation mask of the human leg, combined with the silhouette of the 3D shoe model with a transparent shoe opening, to divide the virtual occlusion area accurately. As for the stabilization operation, since we found that common stabilization and filtering methods, such as Kalman filter, and one Euro filter [5], perform unsatisfactorily in our case. Therefore, we propose a novel method to smooth the trajectory and eliminate jitters.

Since there is no foot benchmark for virtual shoe try-on task so far, we construct and annotate a large-scale pioneer benchmark for training and evaluation. Containing a total of 86K images, all feet in our benchmark are annotated with all virtual shoe try-on-related labels, including the 8 keypoints of each foot, and the segmentation mask of legs and feet. It is worth mentioning that we neither use a multi-camera system for collecting nor the 3D bounding box for annotation. Instead, we propose to capture images of feet with a common monocular camera and annotate data in a novel way. As shown in Fig. 1 (a), trained on our constructed benchmark, our approach achieves a satisfying performance in virtual shoe try-on. Speed evaluations on several widely used smartphones are shown in Fig. 1 (b), the results validate that our approach can bring a smooth virtual try-on effect with a real-time speed of over 30 frames/s (FPS).

The main contributions of this work are four-fold:

- A novel virtual shoe try-on system for smartphones is proposed, which realizes a stable and smooth try-on effect. To our best knowledge, this is the first attempt in the literature to build a practical real-time virtual shoe try-on system.
- A multi-branch network is proposed to realize keypoints detection, 2D pose estimation, and segmentation simultaneously. An origin stabilization method is presented to smooth the trajectory of overlaid shoes and eliminate jitters.
- A large-scale pioneer benchmark containing annotation information of keypoints and segmentation masks is constructed. An origin 6-DoF foot pose annotation method is also presented for efficient annotating.
- Numerous qualitative and quantitative evaluations demonstrate the satisfying realistic effect and high stability of our system. Practical tests on mainstream smartphones validate the high fluency and computational efficiency of ARShoe.

2 RELATED WORK

Note that since few studies focus on virtual shoe try-on so far, we attempt to review the most relevant works here.

2.1 2D Pose Estimation

When applied to the human body, 2D pose estimation refers to the joint coordinates localization, for example, 16 body joints in MPII dataset [1] and 32 body joints in Human3.6M dataset [16]. State-of-the-art (SOTA) 2D pose estimation approaches can be categorized into two types, i.e., heatmap prediction-based and coordinate regression-based methods. Standing as a representative approach of heatmap prediction-based sort, stacked hourglass networks [21] utilizes symmetric hourglass-like network structures to significantly improve the accuracy of pose estimation. Coordinate regression-based methods directly regress the joint coordinates, which has been less studied in recent years owing to lacking spatial and contextual information [4, 10]. There are also related studies on hand pose estimation. For instance, nonparametric structure regularization machine (NSRM) [6] and rotation-invariant mixed graph model network (R-MGMN) [19] perform 2D hand pose estimation through a monocular RGB camera. OpenPose [3] performs 2D pose estimation for multi-person and constructs a human foot keypoint dataset which annotates each foot with 3 keypoints.

2.2 6-DoF Pose Estimation

The common solution for 6-DoF pose estimation is in two-stage. First, predict 2D keypoints, and then estimate 6-DoF pose from the 2D-3D correspondences through the Perspective-n-Point (PnP) algorithm. Attempting to predict the 3D poses of challenging objects, such as partial occlusion, B88 [25] adopts a convolutional neural network (CNN) to estimate the 3D poses based on the 2D projections of the corners of the objects’ 3D bounding boxes. In [29],
the authors also follow a similar procedure. Pixel-wise voting network (PVNet) [23] predicts pixel-wise vectors pointing to the object keypoints and uses these vectors to vote for keypoint locations. This method exhibits well performance for occluded and truncated objects. Dense pose object detector (DPOD) [37] refines the initial poses estimations after a 6-DOF pose computed using 2D-3D correspondences between images and corresponding 3D models. PVN3D [14] extends 2D keypoint approaches, which detects 3D keypoints of objects and derives the 6D pose information using least-squares fitting. In our case, one of the branches in our network is well-trained to predict 2D keypoints in feet, then 6-DoF poses can be obtained utilizing the PnP algorithm.

2.3 Virtual Try-On Techniques

In recent years, the virtual try-on community puts great effort to develop virtual try-on for clothes and achieves promising progress. The main approaches of virtual try-on can be divided into two categories, namely image-based and video-based methods. For the former, VITON [12], and VTNFP [33] generate a synthesized image of a person with a desired clothing item. For the latter, FW-GAN [9] synthesizes the video of virtual clothes try-on based on target images and poses. Virtual try-on for other fashion items has not attracted as much research attention like that for clothes, such as virtual try-on for cosmetics, glasses, and shoes. For glasses try-on, X. Yuan et al. [36] reconstruct 3D eyeglasses from an image and synthesize the eyeglasses on the user’s face in the selfie video. In [38], the virtual eyeglasses model is superimposed on the human face according to the head pose estimation. Chao-Te Chou et al. [7] propose to encode feet and shoes and decode them to the desired image. However, this approach works well in scenes with a simple background and can hardly generalize to practical usage conditions. Moreover, the huge computational cost makes it unsuitable for mobile devices. In contrast, this work focuses on virtual shoe try-on and proposes a neat and novel framework, realizing appeal realistic rendering effect while running in real-time on smartphones.

3 ARSHOE SYSTEM

As shown in Fig. 2, our ARShoe system has three important components: 1) A fast and accurate network for keypoints prediction, pose estimation, and segmentation; 2) A realistic occlusion generation procedure; 3) A 3D model stabilization approach. The foot pose is estimated by the predicted part affinity fields (PAFs) [3], targeting to group estimated keypoints into correct foot instances. Afterward, 6-DoF poses of feet are obtained by mapping 2D keypoints from each foot. Then, the 3D shoe models can be rendered to the corresponding poses. The segmentation results are used to help find the correct area on the 3D shoe model that should be occluded by the leg. Lastly, the 3D model stabilization module ensures a smooth and realistic virtual try-on effect. Moreover, for effective and accurate annotation, an origin 6-DoF foot pose annotation method is also presented in this section.

3.1 Network in ARShoe

Network structure: We devise an encoder-decoder network for the 2D foot keypoints localization and the human leg and foot segmentation. Inspired by [24], the encoder network fuses high-level and low-level features for better representation. The decoder network contains three branches: one to predict the heatmap of foot keypoints, another for PAFs prediction, and the other for human

Figure 2: The working procedure of ARShoe. Consisting of an encoder and a decoder, the multi-branch network is trained to predict keypoints heatmap (heatmap), PAFs heatmap (pafmap), and segmentation results (segmap) simultaneously. Post processes are then performed for a smooth and realistic virtual try-on.

Figure 3: The architecture of the upsample module. It consists of a residual block and two consecutive pixel shuffle layers, which upsamples the 16 × 16 feature map to 64 × 64.
As shown in Fig. 5, each upsample module is composed of a convolution block, a ResBlock, and two pixel shuffle layers. It can generate accurate keypoints, PAFs, and segmentation results while maintaining a relatively low computational cost. To validate the effectiveness of our upsample module, we compare it with the commonly used upsample operation, i.e., bilinear interpolation operation. The results are shown in Fig. 4. It can be seen that our upsample module generates a more accurate estimation.

In our implementation, we set 8 keypoints in the foot as shown in Fig. 5. Therefore, the output shape of the keypoints heatmap is $8 \times 64 \times 64$, corresponding to 8 keypoints respectively. PAFs heatmap have the size of $14 \times 64 \times 64$, as show by the 7 lines in Fig. 5. Each line has two PAFs, with the pixel value represent the projection of the line on x and y axis. The size of the segmentation results is $2 \times 64 \times 64$, with one channel for leg segmentation and another for foot segmentation. To obtain a favorable performance, we adopt $L_2$ loss in all three branches. For more details about the network architecture and the training losses, please kindly refer to supplementary material.

**Keypoints grouping:** There are always two feet in the usage scenarios. Therefore, we predict PAFs to represent the connection between keypoints, which can group the keypoints belonging to the same foot instance. We derive 8 keypoint heatmaps $[H_1, ..., H_8]$ and 8 keypoint coordinates $[k_1, ..., k_8]$ from one image. The corresponding PAFs have 7 elements, which are denoted as $[F_1, ..., F_7]$. As shown in Fig. 5, $k_1$ and $k_3$ are connected. The heatmap of keypoint $k_1$ and $k_3$ predicted by our network is shown in Fig. 6 as $H_1$ and $H_3$, $F_0$ and $F_1$ represent x-direction and y-direction values of the connection relationships of keypoints $k_1$ and $k_3$ respectively, and each PAF has two connecting lines.

By looking for the two points closest to each connecting line, we can connect these two points, as shown in Fig. 6. Keypoints $k_1$ and $k_3$ consist a connection, while keypoints $k_2$ and $k_4$ consist another. That is, keypoints $k_1$ and $k_3$ belong to one foot, and keypoints $k_2$ and $k_4$ belong to another. In this way, all the keypoints are grouped to the corresponding foot instances. As shown in Fig. 7, we divided all keypoints into two groups, which means that the two feet can be distinguished in this way.

6-DoF pose estimation: After keypoints grouping, each group has 8 elements, with one element represents the coordinate of one keypoint. We use the PnP algorithm to generate the $R$ and $T$ of the human foot, according to the 3D points of a standard human foot, camera intrinsics, and the predicted 2D keypoints.

### 3.2 Occlusion Generation

To generate a realistic rendering effect, the 3D shoe model should overlay on an exact position of the input with an appropriate scale and pose. Prior methods rely on depth information [28] or the coordinates of try-on objects in the image [35]. Differently, We utilize the segmentation information of human legs, combined with the silhouette of the 3D shoe model with a transparent shoe opening, to localize the accurate virtual occlusion area.

To produce a realistic effect, we place a transparent 3D object in the area of the shoe opening. The object position can be appropriately moved down to the inside of the shoe opening to retain a certain thickness of the edge of the shoe opening and bring a more realistic feeling. Because the shoe opening is transparent, there
will have two silhouettes after rendering. The outside silhouette is represented as $S_0$, while the inside silhouette is represented as $S_1$.

The occlusion generation process is illustrated in Fig. 8. Firstly, we find the intersections of the leg mask and $S_0$. We then find the corresponding nearest points to $(M_0, M_1)$ in $S_1$, and represent them as $(N_0, N_1)$. Finally, we link $(M_0, N_0)$ and $(M_1, N_1)$ to form a closed region on the mask as a 2D occlusion region and then render the region transparent for virtual occlusion.

### 3.3 3D Model Stabilization

To stabilize the overlaid 3D shoe model, we propose a method that combines the movement of corner points and the estimated 6-DoF foot pose. As we are processing image sequence, we denote each frame as $I_l$, where $l \in \{0, ..., k\}$. Assuming $I_1$ is the current image, thus the previous frame is $I_{l-1}$.

Firstly, we extract FAST corner points of $I_{l-1}$ and $I_l$ in the foot area according to the foot segmentation. These points are represented as $P_{l-1}$ and $P_l$. Then we use bi-direction optical flow to obtain corner points pairs $P_{l-1}' = \{p_{l-1,1}', ..., p_{l-1,n}''\}$ and $P_l' = \{p_{l,1}', ..., p_{l,n}''\}$, which have a matching relationship. Here $n$ is the number of matching points.

The pose of a foot is represented by a $4 \times 4$ Homogeneous matrix, which is from its 3D model coordinate frame to the camera coordinate frame:

$$T = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \in SE(3), \quad R \in SO(3), \quad t \in \mathbb{R}^3$$

where $R$ and $t = \{t^x, t^y, t^z\}$ are the rotation matrix and the translation matrix. The pose of the current frame is $T_1$, while the pose of the previous frame is $T_{l-1}$. The point clouds of the 3D shoe model are denoted as $C$, and the camera intrinsics is $K$:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

where $f_x$ and $f_y$ are the focal length, $c_x$ and $c_y$ are the principal point offset. Because of keypoint detection noise, there are inevitable jitters when using $T_1$ for the virtual shoe model rendering. We leverage $P_{l-1}'$ and $P_l'$ to update a refined pose $T_l''$ for the stabilization.

Firstly, we compute the average displacement $V_{\text{pix}} = (\hat{v}_{\text{pix}}^x, \hat{v}_{\text{pix}}^y)$ of corner points in pixel coordinates.

$$V_{\text{pix}} = \frac{1}{n} \sum_{i=1}^{n} (p_i^l - p_i^{l-1})$$

Secondly, we compute the translation matrix. We assume the depths of foot in the two consecutive frames remain unchanged, i.e., $t^z_l = t^z_{l-1}$. In camera coordinates, the average displacement $V_{\text{cam}}$ of the corner points are:

$$v^x_{\text{cam}} = \frac{\hat{v}_{\text{pix}}^x \times t^z_l}{f_x}$$
$$v^y_{\text{cam}} = \frac{\hat{v}_{\text{pix}}^y \times t^z_l}{f_y}$$

$$V_{\text{cam}} = (v^x_{\text{cam}}, v^y_{\text{cam}}, 0)$$

The translation matrix is:

$$T_l' = [I^t_l + v^x_{\text{cam}} I^r_l, v^y_{\text{cam}} I^r_l] T$$

We project the 3D point clouds $C$ to the two image coordinate systems according to $[R_{l-1} | T_l']$ and $[R_l | T_l]$, which results in $C_1$ and $C_2$:

$$C_1 = K \times [R_{l-1} | T_l'] \times C$$
$$C_2 = K \times [R_l | T_l] \times C$$

The average $L1$ distance between points set $C_1$ and $C_2$ is:

$$D = \|C_1 - C_2\|_1$$

The updating weight $w_{\text{R}}$ of the rotation matrix is:

$$w_{\text{R}} = \alpha \times \ln(D) + \beta$$

Here $\alpha = 0.432$ and $\beta = 2.388$ are empirical parameters. The updating weight $w_T$ of the translation matrix is:

$$w_T = w_{\text{R}} \times w_{\text{R}}$$

Finally, we compute the updated $R''_l$ and $T''_l$ using the following steps:

1. The rotation matrix $R_l$ and $R_{l-1}$ are changed to the quaternions $M_l$ and $M_{l-1}$.
2. The quaternion $M''_l$ is updated by:
   $$M''_l = w_{\text{R}} \times M_l + (1 - w_{\text{R}}) \times M_{l-1}$$
3. The quaternion $M''_l$ is transformed to the rotation matrix $R''_l$.
4. We update the translation matrix at time $l$:
   $$T''_l = w_T \times T_l + (1 - w_T) \times T''_{l-1}$$

Between two consecutive frames, the change of the foot pose is continuous and subtle compared with the irregular and significant noise. To ensure the stability and consistency of the foot pose in the sequence, we first calculate the approximate position of the human foot in 3D space according to Eq. (5). When $T_l$ is updated, the rotation $t$ contributes the most to the reprojection error. After the weighted filtering of the rotation matrix, the change of angle will keep a certain order with the previous frame. Hence, when the change of foot poses across two consecutive frames is slight, the pose of the previous frame will gain more weight than that of the current frame, vise versa.
3.4 6-DoF Pose Annotation

The foot poses information is needed for training the pose estimation branch. Common 6-DoF pose annotation methods include multi-camera labeling [16, 26] and 3D bounding box labeling [11, 27]. The former uses multiple cameras to shoot from different angles. According to the relationship between the cameras, the 3D information of the recovery points can be accurately labeled, but the cost is high, and the operation is complicated. By pulling an outer bounding box, the latter realizes the selection and annotation of objects in a 2D image. However, the bounding box can be hardly labeled with high accuracy.

We designed an innovative annotation method to simulate the rendering process of the final 3D model. Based on the 3D rendering engine, the 3D model of shoes is rotated, translated, and scaled manually to cover the position of shoes in the image, as shown in Fig. 9. Therefore, the final pose transformation matrix $R$ and $T$ of the model can be obtained to achieve accurate annotation. We selected 8 points from the rendered 3D model. The positions of these points can be seen in Fig. 5. Keypoint 1 locates the toe. Keypoint 7 locates the center of the instep position of the foot. Keypoints 3, 4, 5, and 6 are the points that locate the sides of the foot. Keypoints 0 and 2 are the points that locate the heel. Seven lines connect these keypoints. According to the transformation matrix and the camera intrinsics, the 3D points of the model can be projected to 2D point coordinates in the image, which are used as the keypoints for training.

4 EXPERIMENT

In this section, exhaustive experiments are conducted to demonstrate the effectiveness of the proposed system. Firstly, we introduce our experimental settings, including benchmark, implementation details, baselines, and evaluation metrics. Next, we compare the proposed method with SOTA keypoint localization and segmentation approaches in terms of accuracy and speed. Finally, we give the qualitative results to verify the immersive shoe try-on effect.

4.1 Experimental Settings

**Benchmark:** This work constructs the very first large-scale foot benchmark for training and evaluation in virtual shoe try-on tasks. The benchmark contains 86,040 images of the human foot. The annotation includes the keypoints of the foot, and the segmentation mask of human legs and feet. Some scenes with annotation are shown in Fig. 10. The offline data augmentation is performed using rotation, translation, partial occlusion, and background changing.

The augmented dataset contains 3,012,000 images. We use 3,000,000 images for training, and 12,000 images for testing. All images are resized to $256 \times 256$.

**Implementation details:** To train our multi-branch network, we use the Adam optimizer [18] while setting the weight decay to $5 \times 10^{-4}$, and the momentum parameters $B_1 = 0.9, B_2 = 0.999$. The initial learning rate is set to $1 \times 10^{-4}$. In order to speed up the model training, we use distributed training method on PyTorch 1.4 with 4 NVIDIA Tesla P40 GPUs (24GB), and the total batch size is set to 1024. During the training process, we decrease the learning rate by 10% at epochs 20, 30, and 60, respectively. The entire network is trained for a total of 120 epochs.

**Baselines:** To evaluate the performance of ARShoe in different subtasks, we compare its performance of pose estimation and foot segmentation with other SOTA approaches. In our shoe try-on pipeline, the keypoint localization is similar to human pose estimation. Therefore, we compare it with the SOTA human pose estimation methods, including Lightweight OpenPose [22] and Fast Pose Distillation (FPD) [39]. For the segmentation performance, we compare our approach with YOLACT [2] and SOLOv2 [32]. Networks adopting different backbones are evaluated, i.e., Darknet1, Resnet [13], and DCN [8] with FPN [20].

**Remark 1:** There is no open-source project about virtual shoe try-on to our best knowledge, thus comparisons are performed among ARShoe and SOTA approaches of corresponding subtasks.

1https://pjreddie.com/darknet/
4.2 Computation Cost and Speed Evaluation

The amount of calculation, model parameters, and processing speed are evaluated exhaustively to demonstrate the efficiency of the proposed system. The involved experimental platforms include an NVIDIA Tesla P40 GPU, an Intel Xeon CPU E5-2640 v3 (2.60GHz) CPU, and different types of smartphones.

**Amount of calculation and parameters:** We compare our network with SOTA pose estimation methods and segmentation methods in terms of calculation and parameters. As shown in Table 1, ARShoe has the fewest Flops (0.984 G) and parameters (1.292 M), while the second lightest approach Lightweight OpenPose still holds the Flops of 4.327 G and the parameters of 4.092 M. For segmentation methods, SOLOv2-Resnet50-FPN is the second lightest approach, which still has 6.3 times more Flops and 20.4 times more parameters than ARShoe. One can draw that the amount of calculation and parameters of our network is very small compared with other SOTA methods.

**Remark 2:** This is not a fair comparison for ARShoe, since our network can deal with both pose estimation and segmentation tasks simultaneously, while other involved approaches are designed for one single task. Even so, our approach still shows much fewer computational needs and parameters.

**Processing speed on GPU and CPU:** The processing speed of the methods on different devices is recorded. As shown in Table 1, ARShoe runs efficiently on both GPU and CPU comparing to other pose estimation and segmentation approaches. Specifically, the inference time of ARShoe on a single image is 11.844 ms on GPU and 59.919 ms on CPU.

**Processing speed on smartphones:** The above two comparisons both demonstrate the computation efficiency and light-weight of ARShoe. Therefore, we further implement the proposed approach on mobile devices to validate its practicability. Four common smartphones are adopted in this evaluation, respectively iPhone 8p, iPhone 11, OnePlus 5T, and Huawei P30. The whole architecture can be divided to three main processes, i.e., network forwarding, pose estimation and stabilization, and rendering and occlusion generation. Table 2 reports the inference time of each processes. The speed of the whole virtual try-on system on different smartphones is also presented in Fig. 1 (b) by FPS. We can see that ARShoe realizes a real-time speed of over 30 FPS on all four smartphones. On iPhone 11, ARShoe even reaches a speed of nearly 60 FPS. The promising results validate that ARShoe is suitable for smartphones.

**Remark 3:** Our system executes these three processes in parallel, therefore the overall frame rate is determined by the process that takes the longest time.

4.3 Performance Evaluation

**Keypoints localization task:** The comparison of different keypoints localization methods on the test set is shown in Table 3. The GPU speed of each approach is also reported. For a fair comparison, the involved approaches are all trained on our collected training dataset. It can be seen that with a promising real-time speed, ARShoe achieves competitive keypoints estimation performance. Comparing to Fast-human-pose-teacher [39] with the best mAP, ARShoe is only 0.041 lower, while is 5.5 times faster, realizing a favorable balance of speed and accuracy.

**Segmentation task:** We compare our method against two SOTA segmentation methods, i.e., YOLACT and SOLOv2, adopting different backbones. The performance comparison and GPU speed of each method are presented in Table 4. Running at a fast speed of

### Table 1: Comparison of computation cost and speed.

| Method                     | Flops (G) | Parameters (M) | GPU time (ms) | CPU time (ms) |
|----------------------------|-----------|----------------|---------------|---------------|
| Lightweight OpenPose       | 4.327     | 4.092          | 14.689        | 79.433        |
| Fast-human-pose-teacher   | 20.988    | 63.595         | 64.960        | 212.880       |
| Fast-human-pose-student   | 10.244    | 28.536         | 61.204        | 162.264       |
| Fast-human-pose-student-FPD | 10.244   | 28.536         | 61.204        | 162.264       |
| YOLACT-Darknet53           | 16.416    | 47.763         | 29.660        | 1035.180      |
| YOLACT-Resnet50            | 12.579    | 31.144         | 26.395        | 1116.825      |
| YOLACT-Resnet101           | 17.441    | 50.137         | 40.254        | 1876.076      |
| SOLOv2-Resnet50-FPN        | 6.161     | 26.412         | 23.513        | 893.511       |
| SOLOv2-Resnet101-FPN       | 10.081    | 45.788         | 35.411        | 1041.213      |
| SOLOv2-DCN101-FPN          | 14.334    | 55.311         | 40.211        | 1322.024      |
| ARShoe                     | 0.984     | 1.292          | 11.844        | 59.919        |

### Table 2: Speed analysis of ARShoe on different smartphones (ms).

| iPhone 8p | iPhone 11 | OnePlus 5T | Huawei P30 |
|-----------|-----------|------------|------------|
| Network forwarding | 27.278   | 4.338      | 24.037     | 25.994    |
| Pose estimation and stabilization | 13.715  | 11.226     | 28.729     | 21.578    |
| Rendering and occlusion generation | 21.636  | 17.122     | 24.326     | 28.813    |
| Overall speed | 27.278 | 17.122     | 28.729     | 28.813    |

### Table 3: Comparative results of foot keypoints localization.

| Method                        | mAP  | AP50 | AP75 | AP90 | Speed (FPS) |
|-------------------------------|------|------|------|------|-------------|
| Lightweight OpenPose [22]     | 0.719| 0.958| 0.788| 0.399| 68.18       |
| Fast-human-pose-teacher [39]  | 0.817| 0.976| 0.884| 0.617| 15.39       |
| Fast-human-pose-student [39]  | 0.786| 0.974| 0.857| 0.490| 16.33       |
| Fast-human-pose-student-FPD [39]| 0.797| 0.976| 0.863| 0.524| 16.33       |
| ARShoe                        | 0.776| 0.972| 0.855| 0.490| 84.43       |

### Table 4: Comparative results of foot and leg segmentation.

| Method       | Backbone | mIoU     | AP  | Speed (FPS) |
|--------------|----------|----------|-----|-------------|
| YOLACT [2]   | Darknet53| 0.895     | 0.921| 33.72       |
|              | Resnet50 | 0.890     | 0.918| 37.60       |
|              | Resnet101| 0.903     | 0.928| 24.85       |
| SOLOv2 [32]  | Resnet50-FPN| 0.925| 0.943| 42.90       |
|              | Resnet101-FPN| 0.939| 0.952| 28.24       |
|              | DCN101-FPN| 0.943     | 0.960| 24.87       |
| ARShoe       | None     | 0.901     | 0.927| 84.43       |
Figure 11: Virtual try-on results obtained by ARShoe. The second and fourth rows show the input images, while the first and third rows show the virtual try-on effects. ARShoe is able to handle feet in various poses and different environments, generating a realistic try-on effect.

84.43 FPS on GPU, ARShoe can still achieve a promising segmentation performance, which is not inferior to the SOTA approaches designed specifically for segmentation. The mIoU and AP of ARShoe are 0.901 and 0.927, which are 0.042 and 0.033 lower than the first place, respectively. Considering the practicability in mobile devices, our approach realizes the best balance of computation efficiency and accuracy.

Remark 4: Since this work targets real-world industrial applications, we argue that the slight margin in performance is insignificant compared to the high practicability that ARShoe brings.

6-DoF pose estimation performance: We also evaluate the performance of our 6-DoF pose estimation method on the test set. The average error of three Euler angles is 6.625°, and the average distance error is only 0.384 cm. Such satisfying 6-DoF pose estimation performance demonstrates that ARShoe can precisely handle the virtual try-on task in real-world scenarios.

Qualitative Evaluation: Some virtual shoe try-on results generated by ARShoe are shown in Fig. 11. It can be seen that our system is robust to handle various conditions of different foot poses and can realize a realistic virtual try-on effect in practical scenes. A virtual try-on video generated by ARShoe is also provided in the supplementary material.

5 CONCLUSION
This work presents a high efficient virtual shoe try-on system namely ARShoe. A novel multi-branch network is proposed to estimate keypoints, PAFs, and segmentation masks simultaneously and effectively. To realize a smooth and realistic try-on effect, an origin 3D shoe overlay generation approach and a novel stabilization method are proposed. Exhaustive quantitative and qualitative evaluations demonstrate the superior virtual try-on effect generated by ARShoe, with a real-time speed on smartphones. Apart from methodology, this work constructs the very first large-scale foot benchmark with all virtual shoe try-on task-related information annotated. To sum up, we strongly believe both the novel ARShoe virtual shoe try-on system and the well-constructed foot benchmark will significantly contribute to the community of virtual try-on.

6 ACKNOWLEDGMENTS
This work was supported by grants from the National Key Research and Development Program of China (Grant No. 2020YFC2906200).
SUPPLEMENTARY MATERIAL

A ABSTRACT
This supplementary material presents the detailed network structure, parameter settings, and training losses of ARShoe.

B DETAILS OF NETWORK ARCHITECTURE
Figure 12 illustrates the detailed network architecture and parameter settings of our multi-branch network. Firstly, the encoder part extracts low-level features utilizing a convolution block and two depthwise separable convolution blocks (DSConv). The high-level features are then obtained through three residual bottleneck blocks (Bottleneck), a pyramid pooling block, and a depthwise convolution block (DWConv). Afterward, both low-level and high-level features go through a convolution block to unify their channel numbers. To integrate high-level semantic features and low-level shallow features for better image representation, high-level and low-level features are summated and go through an activation function ReLU. In the decoding process, three upsample modules are adopted to decode the fused features to keypoint heatmap (heatmap), PAFs heatmap (pafmap), and segmentation results (segmap). The dimension variation process of feature maps is presented in Fig. 12.

C TRAINING LOSSES
We utilize $L_2$ loss for all three subtasks in our network. The ground truths of the keypoints prediction branch, the PAFs estimation branch, and the segmentation branch are the annotated keypoints, connections between keypoints, and segmentation of foot and leg, respectively. Denoting losses of these three branches as $L\text{heatmap}$, $L\text{pafmap}$, and $L\text{segmap}$, respectively, the overall loss $L$ of our network can be formulated as:

$$L = \lambda_1 L\text{heatmap} + \lambda_2 L\text{pafmap} + \lambda_3 L\text{segmap}$$  \hspace{1cm} (12)

where $\lambda_1, \lambda_2,$ and $\lambda_3$ are coefficients to balance the contributions of each loss. In our implementation, they are set to 2, 1, and 0.1, respectively.

Figure 12: The detailed network architecture of ARShoe.